UNIX

최신지도서

평양콤퓨터기술대학
외국문도서출판사
주체91

UNIX
최신지도서

평양콤퓨터기술대학

외국문도서출판사

차 례

제 5 장. GNU emacs편집기

제 6 장. 파일체계

제 7 장. 파일속성

제 8 장. 쉘

제 9 장. 려과기

제 10 장. 프로쎄스

제 11 장. TCP/IP망작업도구

제 12 장. X Window체계

제 13 장. 전자우편

제 14 장. 인터네트

제 15 장. 정규식을 리용한 려과기 grep와 sed

제 16 장. awk를 리용한 프로그람작성

제 17 장. 환경의 전용화

제 18 장. 쉘프로그람작성

머 리 말

UNIX는 1978년에 켄 톰슨과 데니스 리치에에 의하여 《고도의 이식가능성》을 제공해 주는 《다목적》조작체계로서 세상에 소개되였다. 그때로부터 UNIX는 계속 발전의 길을 걸어 왔으며 오늘날 그 지위가 확고히 다져 져 현재는 중요한 작업들에 쓰이고 있다. UNIX는 실험실에서의 콤퓨터지원설계나 모의실험에 리용되여 그 개발을 돕는 가동환경으로 되고 있다. UNIX는 C언어를 《탄생》시켰다. UNIX에서는 거대한 규모의 자료기지응용프로그람들이 실행되고 있다. 오늘날 전자상업은 UNIX에 의하여 보장되고 있다.

하지만 UNIX가 유명해 지기까지의 길이 언제나 순탄한것은 아니였다. UNIX에는 지지자와 시비군이 언제나 함께 뒤따르군 하였다. 초학자들은 UNIX의 원리를 습득하기 매우 어려워 하고 있으며 무엇때문에 UNIX가 Windows가동환경과 완전히 다른 형식으로 작용하는지 리해되지 않아 하고 있다. 경험 있는 콤퓨터전문가들까지도 해독하기 어려운 도구들과 문법으로 엮어 진 이 수수께끼 같은 세계에 대해 자기들이 무기력해 지는 감을 느끼고 있다. 그러나 Windows는 이와는 전혀 다르다. 그러면 왜 UNIX가 이런 식으로 설계되였겠는가?

여기에는 그럴만한 사연이 있다. UNIX는 세계를 상대로 하여 설계된것이 아니다. 어느 한 프로그람작성자들의 단체는 자기들의 프로그람들을 실행시키기 위해 UNIX를 만들어 내게 되였는데 그들의 프로그람들은 다른 누구에게도 소용되지 않는것이였다. 그런데 판매를 위한 적극적인 노력이 없이도 계속 광범한 호평을 받고 있다는 사실은 UNIX가 자기의 고유한 위력을 가지고 있다는데 대해 시사해 주고 있다. UNIX는 자기의 규격들을 개방하는것을 원칙으로 하고 있기때문에 모든 하드웨어에 공개되여 있으며 현재는 인터네트에 제공되고 있다.

UNIX는 얼마 안되는 단순한 착상들에 기초하고 있지만 이전에는 볼수 없었던 많은 특징들을 가지고 있다. UNIX는 조작체계란 무엇이며 왜 조작체계에 정통해야 하는가에 대해 다시 규정해 주었다. UNIX《지령행》은 무수한 선택항목들과 복잡한 문법들을 가지고 있는것으로 하여 많은 사람들의 실망을 자아내게 되였다. 하지만 그의 기능들과 결합적으로 작용하는 지령들의 능력으로 하여 UNIX는 명백하게 그 무엇도 대신할수 없는것으로 간주되고 있다.

UNIX의 모든 특징들에 대해 다 소개하기에는 교과서 한권이 너무나도 부족하지만 어쨌든 그 방대한 량가운데서 특별히 주의를 돌려야 할 중요한 개념들을 골라 묶어 주려고 하였다.

이 책은 프로그람작성법과 조작체계들, 체계관리와 관련되는 갖가지 UNIX의 기본원리들을 습득하는데서 매우 효과적인 교재로 될것이다. 어떤 경우에도 한개 조작체계에 대한 지식을 가지는것은 매우 유익하다. 왜냐하면 하나의 프로그람작성언어를 소유한 사람일수록 쉘의 프로그람작성특징들에 대한 리해를 더 잘할수 있기때문이다.

오늘날 왜 UNIX를 알아야 하는가

UNIX는 4개의 주요단계들을 거쳐 성장해 왔다. 초기에는 과학기술연구소들을 위한 제품으로 간주되였었다. 이와 같은 사업을 떠나서는 UNIX가 존재할수 없었다. UNIX가 《불친절한》것으로 하여 과학기술분야를 제외한 다른 분야들에서는 그리 쓸모가 없었던것이다. 지금에 와서 우리가 인식하게 되는것은 UNIX가 바로 이 대학과 연구소들의 개발연구결과라는것이다.

다음단계는 UNIX가 기업체들과 정부기관들에 쓰이기 시작한것이다. 거대한 UNIX는 자료기지작업을 위한 조작체계로 선택되였다. 독자들이 훌륭한 자료기지관리자가 되기를 원한다면 UNIX를 알아야 한다.

UNIX성장의 세번째(가장 중요한) 단계는 인터네트이다. 망과 Web에서 보게 되는 많은것들이

본래는 UNIX공동체가 진행한 어렵고 적극적인 작업의 덕분이다. TCP/IP가 제일 먼저 이식된 곳도 UNIX체계였다. 대부분의 망봉사기들은 UNIX기계들이다. 인터네트봉사제공자들(Internet Service Providers)은 UNIX기계를 사용하고 있다. Web에서 나타나는 모든 양식들을 뒤에서 조작하고 있는 perl은 UNIX의 산물이다. 간단히 말해서 인터네트는 곧 UNIX이며 인터네트작업에 대해 인식하고 지역인터네트기술을 개발하자면 UNIX를 알아야 한다.

Linux에 대한 관심이 조용히 그러면서도 비상히 커가고 있는것 역시 최근시기의 추세이다. Linux는 새 세대 대학생들을 매혹시켰고 지금은 경제분야에 깊이 침투되고 있는 자유로운 UNIX이다. Linux는 풍부한 인터네트관련도구들을 가지고 있는것으로 하여 광범한 호평을 받고 있다. Linux의 앞으로의 전망은 대단히 크며 이로부터 여기서는 Linux의 주요특징들도 서술하고 있다. 비록 Linux가 본래의 AT&T코드를 전혀 사용하지 않고 있지만 《정신》에 있어서나 원리에 있어서 UNIX와 꼭 같다.

실지로 왜 UNIX를 좋아 하는가

최근 시기 UNIX를 사용하지 않고 있는듯 하지만 사실 많은 사람들이 《뒤문》을 통해 UNIX에 들어 가고 있으며 그 뒤문이란 X Window체계로서 UNIX기계들에 제공되는 도형창문체계를 말한다. 이런 사람들은 차림표에 기초한 GUI응용프로그람들을 가지고 체계를 구성하고 있으며 지어는 파일들의 이름이나 기입내용들에 대해서는 전혀 주의를 돌리지 않고 있다. 그러나 이런 식으로는 UNIX에 대해 절대로 알수 없다. 그들은 공상에서 깨여 나 얼마 못가서 UNIX시비군들의 무리에 끼여 들게 될것이다.

사실 UNIX는 매혹적인것이다. 이 체계는 특정한 일감을 수행하는 지령과 프로그람들에 기초하고 있다. 지령들은 선택항목을 사용하며 매개의 선택항목은 지령들이 제각기 다르게 작용하게 한다. 이 지령들은 대체로 복잡한 과제들을 수행하기 위해 서로(려파기로) 결합될수 있다. 그것들중 일부는 본문을 배치하거나 변경시키는데 사용되는 위력한 패턴정합기능(정규식들이라고 함)까지도 사용한다. 이 모든것에는 스크립트언어가 첨부되며 사용자는 이 체계로 할수 있는 모든것을 무제한 설정할수 있는 도구묶음을 가지게 된다.

UNIX지령들은 일반적으로 대화식이 아니므로 자동화체계개발에 적합하다. 창문환경에서는 쉽게 할수 없는 일들을 UNIX에서는 얼마든지 할수 있다. Windows사용자에게 천개의 파일들을 BMP로부터 GIF형태로 변환시키라고 요구하게 되면 그는 즉석에서 깜짝 놀랄것이다. 사실 이것은 UNIX에 대해 잘 알기만 하면 단 3~4개의 간단한 코드행들을 써서 수행할수 있는 문제이다. 마우스를 천번(실지는 그이상)씩이나 누를 필요가 없다.

UNIX의 많은 능력들이 감추어 져 있다는 사실은 흥분을 불러 일으킨다. 한개의 그릇에 모든것을 다 담지는 못하는 법이다. UNIX는 사용자가 부단히 창조하고 혁신할것을 권고하고 있다. 지령들을 결합하여 쓰거나 복잡한 일감을 수행하는 스크립트를 설계하는것은 UNIX사용자들에게 있어서 실제적인 도전으로 된다. 이것이 바로 UNIX이며 UNIX는 이와 같은 식으로 남아 있게 될것이다. 이 점을 옳게 인식하게 되면 독자들은 정확한 길에 들어 서게 되며 이 책은 쓸모 있는것으로 된다.

이 책은 어떻게 구성되여 있는가

UNIX판본은 크게 2개의 서로 다른 부류들로 나눌수 있는데 하나는 AT&T벨연구소의 System V이고 다른 하나는 캘리포니아종합대학의 버클리판본이다. 보다 정확히 말한다면 SVR4와 BSD UNIX이다. 이 책은 Solaris의 UNIX기능들에 대한 사용을 주시하면서 UNIX를 총체적

으로 묘사하려고 하였다. Linux도 BSD에 기초하고 있으므로 이 책에서는 그에 대한 설명을 함께 주고 있다.

주제설정은 다른 교재들과 다르게 구성되여 있다. 여기서는 체계구성내용이 여러개의 장들에 펼쳐 진것을 자주 보게 되며 서로 류사한 자료들은 리해와 참고에 편리하게 한데 묶어 놓았다. 이와 같은 구성체계를 한번 리해하게 되면 기능이나 지령을 찾아 내는것이 어렵지 않을것이다.

주제	장
편집기들	4, 5
파일체계	6, 7, 21
쉘	8, 17, 18, 19
프로쎄스	10, 22
TCP/IP와 인터네트	11, 13, 14, 23, 24
정규식	4, 5, 15, 16, 20
려파기	9, 15, 16, 20
체계와 망관리	21, 22, 23, 24

지령이나 기능을 이 책의 장들에 배치하는데는 일정한 원칙이 있다. 파일속성과 관련한 장이 따로 있는것을 아마 다른 교재들에서는 보지 못했을것이다. 쉘은 3개의 특수한 형식들인 해석기(제8장), 환경전용화기(제17장), 프로그람작성언어(제18, 19장)와 관련한 부분들에서 나타나게 된다. 이 책에서는 일부 사람들처럼 한개 장에서 쉘프로그람작성과 일감조종에 대해 다같이 론의하는것과 같은 과오를 범하지 않았다. 일감조종은 비록 쉘의 특징이지만 프로쎄스범주에 속해야 한다고 본다. 때문에 제10장에서는 프로쎄스들을 만들고 제거하며 그 일정을 작성하는것도 보게 된다.

이 책의 특징은 무엇인가

우리는 이 책을 의식적으로 다른 책들과 구별되게 만들려고 하지는 않았다. 물론 조작체계에 대해서 일정하게 알고 있었다고는 하지만 콤퓨터앞에서 UNIX와의 대면은 이번이 처음이였으며 우리에게는 의지할 곳도, 지도해 줄 사람도 없었다. 이와 같이 《한지상태》에 처한데다가 더우기 난점으로 제기된것은 UNIX의 내용들이 실지 상상했던것과는 다르게 느껴 지는것이였다. 그리하여 UNIX에 대해 다른 사람들이 설명한 방법을 정말이지 접수할수 없었으며 독자들이 언제나 가지고 다니면서 애용할수 있는 《진짜》 UNIX책에 대해 생각하게 되였다. 이러한 구상을 반영한것으로 하여 이 책은 다른 책들과 구별되는 특징을 가지게 되였다. 아래에 그 중요한 5가지를 하나씩 서술한다.

1. 표현의 명쾌함

UNIX개념들은 때로 추상적이며 그렇지 않은 경우에도 실지세계와의 관계가 옳게 평가되지 못하는 경우가 드문하다. 매 개념들은 자기가 목적한 내용을 정확히 나타낼수 있게 분석되여야 한다고 본다. 왜 특징에 대해 먼저 인식해야 하는가, 그것이 어디에 적용되는가, 표준설명이 정확한가, 실례들은 언제나 인식의 애매한 부분뒤에 놓여야 하는가와 같은 질문들에 정확한 답을 주지 못한다면 혼란과 애매한 상태에 놓이게 된다는것은 불 보듯 명백하다 .

이 책에서는 매개의 개념들이 적당한 방법으로 설명되고 있다. 실례로 UNIX체계의 기초원리들 가운데서 표준출력에 대해 보기로 하자. 표준출력이 무엇인가에 대해서는 알고 있지만 우리는 다른 책들에서 이에 대해 설명하는 방법에 대해서는 공감이 가지 않는다. 그러나 지령렬 who>newfile을 아래와 같이 서술하면 더 잘 인식하리라고 본다(220페지).

쉘은 >를 보고 표준출력의 방향이 절환되여야 한다는데 대해 인식하며 newfile을 열고 그안에 흐름을 쓰며 그다음에는 파일을 닫는다. 이 모든 작업은 who가 이에 대해 전혀 모르는 상태에서 진행된다.

지령이 자기의 입출력자원에 대해 전혀 모르고 있다는 사실을 아는 사람이 몇명이나 되는가? 그리고 wc foo에 앞서 wc<foo의 사용을 선택할 때 여기에 중요한 의미가 담겨 져 있다는 사실을 아는 사람은 몇명이나 되는가(220페지)? 지령들이 자기의 입력자원에 대해 전혀 모르는 상태에서 동작하게 해야 되는 필요가 언제 생기게 되는가(226페지)? echo "Enter your name \ c"에 있는 \ 이 실지로 문자 c의 원래의미를 제거하는가(218페지)?

독자들은 자기가 쉘에 대해 리해했다고 만족해 하기전에 우선 우의 질문들에 대한 대답을 알아야 한다.

이밖에도 독자들이 정확히 인식해야 할 많은 내용들이 아직까지도 표준도서들에서 정확히 서술되지 못하고 있다. 독자들은 이 모든(보다 더 많은) 질문들에 대한 대답을 이 책에서 찾게 될것이다.

2. 기초적인것과 종합적인것.

어느 한 고찰자는(이 책을 평하기에 앞서) 대학의 한 학기교재로서 UNIX책을 쓰는데서 골치거리는 대학생들에게 충분한 기초내용을 주는 동시에 그들의 수준이 올라 가는데 따라 사용할수 있는 충분한 종합지식과 《참고》내용들을 담아야 하는것이라고 하였다. 이와 같은 두개의 조건에 꼭같이 맞아 떨어 지는 책을 만드는것이 바로 우리가 부닥친 가장 어려운 과제였다. 하지만 이제는 이 과제가 수행된셈이다. 물론 그에 대한 최종평가는 독자들이 해야 할것이다.

이 책에는 방대한 량의 자료들이 하나의 구성체계로 펼쳐 져 있는데 실지 책의 규모보다 훨씬 더 많은 량을 담고 있다. 우리는 모든 지식들을 보다 알기 쉽게 보여 주기 위해 노력하면서도 쓸데없는 군말이 전혀 없게 하였다. 그리고 자료를 한데 묶어 놓는 기본방식에서 벗어 나지 않으면서도 선진자료들은 장의 마지막부분들에 배치하는 방법으로 기본내용들과 분리시켜 놓았다. 초학자들은 초기과정안에서 이 부분을 간단히 무시해 버리면 된다.

이 책은 또한 종합지도서이다. 중요한 지령들에 대해서는 표에 선택항목들을 정의하였으며 vi와 emacs편집기들과 관련한 장들에는 많은 표들이 있다. 편집기사용법에 대해 한번만 알게 되면 다른것을 아는데 도움이 되는(vi와 emacs의 특징들을 비교한) 부록 2를 보시오. C쉘은 함수들을 사용하는가? Korn쉘과 bash에서 command라고 이름 지은 지령의 역할은 무엇인가? 4개의 쉘모두를 상세히 비교한 부록 4를 보시오. 정규식들은 주로 려파기들에 의해 서로 다르게 사용된다. 정규식모형은 부록 3이라는 《한 장소》에 다 서술되여 있다.

책의 마지막에는 용어해설을 주었으며 색인에도 특별한 관심을 돌렸다.

3. 생동한 그림들

최근 독자들은 UNIX책들에서 많은 그림들을 보게 된다. 하나의 개념을 설명하는데 실지 몇개의 그림이 요구되는가? 우리는 일부 저자들처럼 많은 그림을 리용하지 않고 그대신 그림을 보다 효과적으로 그리려고 애 썼다. 실례로 그림 8-2에서는 표준출력에 대해 다른 방법들보다는 우월하게 보여 주고 있다. 아마 표준출력을 이런 식으로 묘사한것을 본적이 없을것이다. 하지만 이 그림은 많은 의문들에 대답을 주고 있다. 독자들은 표준출력이 3개의 목적지를 가질수 있다고 생각해 본적이 있는가? 이 그림은 천만마디의 말을 손 쉽게 대신해 주고 있다. 그림 8-1을 보게 되면 쉘이 작용하는 형태의 복잡성이 저절로 풀리게 된다. 그림 10-2에서는 프로쎄스제거기구에 대하여 전파탐지기, 포, 비행기에 비유하여 그림으로써 리해하기 쉽게 하였다. 또한 그림 13-7에서는 우편통신원을 출현시켜 SMTP와 POP가 어떻게 전자우편을 처리하는가에 대해 아주 의미심장하게 보여 주고

있다.

4. 보충자료에 의한 세부인식

교재에서 이미 언급된 문제도 그것이 중요한것이면 독자에게 다시 상기시킬 필요가 있다. 사실 교육학적으로 잘 타산된 보충자료들이 들어 있는것이 이 책의 주요특징의 하나이다. 이 책에는 약 400개이상의 보충자료들이 들어 있다. 이와 같은 보충자료들은 주해, 주의, 참고와 같이 여러가지 명칭으로 씌여 있다.

이 책은 원래 Linux에 대한 책은 아니다. 하지만 Linux는 UNIX계렬의 중요성원으로 볼수 있다. 일반적으로 Linux지령들은 보다 많은 선택항목들을 가지고 있으며 그들중 일부는 그야말로 훌륭하다. 이 책에서는 Linux의 특이한 기능들과 Linux에 의해 다르게 처리되는 특징들에 대해 특별히 강조하였다. 이에 대한 실례들은 찾아 보기 쉽게 배치하였다. 펭긴새표식이 있는 대목을 찾아보기 바란다.

이 책에서는 Bourn쉘을 《기초》쉘로서 사용하였지만 C쉘, Korn쉘, bash와 같은 다른 쉘들에 대하여서도 론의하였다. 이 쉘들을 위한 장을 따로따로 설정하지 않고 먼저 개념들에 대한 일반론의를 진행하고 따로 설정한 부분에서 강조해 주었다.

5. 물음과 련습

이 책에서는 독자의 지식상태를 시험해 보기 위해 많은 물음들을 제기하였는데 그 수는 무려 900여개가 넘는다. 3분의 1이상은 매장의 뒤에 있는 시험문제들에 있으며 그 답은 부록 8에 주어 져 있다. 이 물음들은 모두 초학자들을 위주로 한것들로서 독자들은 매번 다음장으로 넘어 가기전에 여기에 대한 대답을 알고 있어야 한다. 련습문제에는 보다 엄격하고 폭 넓은 물음들이 있다. 일부 물음들은 정말 도전적이여서 쩔쩔 매게 하며 상당한 시간이 걸려야 풀수 있는것들이다. 이 련습문제들은 UNIX에 대한 독자들의 지식을 보충해 주고 더 풍부히 해주므로 절대로 무시하지 말아야 한다.

이 책의 모든 실례들은 UNIX와 Linux의 많은 체계들에서 검증된것이지만 사실 그것들이 모든 체계에서 실수없이 실행되리라고는 담보할수 없다. UNIX의 분화는 그의 일반화를 사실상 불가능하게 만들었다. 일부 지령들이 사용자의 체계에서 사용할수 없거나 전혀 왕청 같은 통보문들을 사용자의 체계에 보내올수도 있다. 사용자는 이와 같은 단계를 뛰여 넘을줄 알아야 하며 체계에 오유가 있다고 무턱대고 속단하지 말아야 한다.

결속에 앞서 한가지 간단한 주의를 주려고 한다. 많은 사람들이 한때의 잘못된 생각으로 UNIX《뻐스》들을 놓치고 지금 후회하고 있다. 그들은 모두 바로 마우스에 반했지만 마우스가 결코 만능으로 될수는 없다. 독자들에게서는 이와 같은 일이 일어 나지 않기를 바란다. 물론 독자들이 UNIX를 전혀 바라지 않거나 더이상 필요가 없게 되였다면 문제는 다르다. 이 체계의 도구들에 대해 배워야 하며 새로운것을 창안하느라 하지 말고 자기가 배운 도구들에 의거하는것이 좋다. 열정과 자신심을 가지고 이 책을 읽기 바란다.

제 1 장. 첫 걸음

이 장에서는 UNIX세계에 대한 가장 기초적인 내용들을 취급한다. 여기서 우리는 콤퓨터에 왜 조작체계가 필요하며 UNIX가 어떻게 그 요구를 만족시켜 주고 있는가에 대하여 배우게 된다. 우리는 두 차례의 실천과정을 통하여 UNIX체계를 운영하는 방법을 배우게 된다. 또한 실지체험을 통하여 UNIX지령들에 대해서와 파일 및 등록부처리방법에 대한 지식을 얻게 된다.

이 지식을 습득하고 나면 UNIX가 걸어 온 거치른 리면세계에 대하여 알게 될것이다. 우리는 또한 각이한 원천들에 의하여 UNIX가 어떻게 풍부해 졌으며 어떻게 분화되였는가에 대하여 알게 될것이다. 또한 여기서 배우게 되는 설계와 관련된 지식은 UNIX가 무엇때문에 때때로 외관상 서투른 작용을 하는가를 리해하는데 도움을 주게 될것이다.

이밖에도 UNIX의 기능들에 대하여 보게 되며 그 기능들사이에서 두 매개물들이 체계의 모든 작업들을 어떻게 처리하는가에 대해서도 보게 될것이다. 여기서 부닥치게 되는 일부 개념들은 비록 추상적인것처럼 보일수 있으나 초기에 리해를 못한다고 하여 단념해서는 안되며 습득할수 있는껏 습득하고 다음 장들로 넘어 가도록 해야 한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 콤퓨터에 왜 조작체계가 필요한가에 대하여 리해한다(1.1).
- UNIX에서의 블로크구성방법이 사용자로 하여금 어떻게 도구들을 만들수 있게 해주는가에 대하여 리해한다(1.2).
- 건반에 있는 특수한 문자들의 위치와 기능에 대하여 인식한다(1.4).
- UNIX체계에 가입하거나 탈퇴하는 방법을 배운다(1.6).
- passwd, who, tty, set, echo지령들의 사용법을 배운다(1.7).
- SHELL과 TERM변수들을 평가한다(1.7).
- 무엇이 잘못되였으며 그 문제들을 처리하기 위하여 특수건들을 어떻게 사용하는가에 대해 배운다(1.9).
- mkdir, ls, cd, pwd지령들을 사용하여 등록부들을 관리한다(1.10).
- echo, cat, wc지령들로 파일의 단어들을 만들고 현시하며 계수한다(1.10).
- UNIX가 어떻게 하나의 원천으로부터 시작되였으며 다른 변종들로 분화되였는가를 인식한다(1.11).
- 강력하고 생활력 있으며 자유롭게 선택할수 있는 Linux의 출현에 대하여 배운다(1.12).
- UNIX를 특징 지어 주는 개념들과 기능들을 배운다(1.13).

1.1 조작체계

독자들은 하드웨어와 쏘프트웨어에 대해 들은적이 있을것이다. 하지만 조작체계가 무엇인지 의문을 가져 본적이 있는가? 혹은 무엇때문에 조작체계 하나에 대해서만 서술하는데 옹근 하나의 책을 써야 하

는가에 대해 의문을 가져 보았는가? 독자들은 아마 문서편집프로그람과 같은것에 대해서는 충분한 지식을 가지고 있을것이다. 사실 우리는 속도를 바꿀 때마다 무슨 일이 일어 나겠는지 정확히 알려고 하지 않고 자동차를 리용하고 있는셈이다. 대상이 제대로 동작하고 있는데 무엇때문에 그 대상의 속내막에 대해 알려고 애를 쓰겠는가?

문서편집프로그람을 단지 리용하기만 하는 경우에는 이러한 론의가 크게 의의를 가지지 못할것이다. 그러나 이러한 응용프로그람들을 작성하려고 한다면 사용자가 준 지시를 콤퓨터가 어떻게 처리하는가에 대해 알아야 한다. 만일 독자가 체계관리자로 되기를 희망한다면 하드디스크에서 파일을 어떻게 구성하며 사용자의 령역을 다른 사용자가 침범하지 못하게 담보하자면 어떻게 해야 하는가에 대해 알아야 한다. 사용자는 자기 콤퓨터의 조작체계를 알고 있어야 한다.

조작체계(OS)는 콤퓨터에 《생명》을 주며 콤퓨터가 작업할수 있는 기초정보를 제공해 준다. 기술적으로 말한다면 조작체계는 체계가 시동될 때 콤퓨터의 주기억장치에 적재되는 프로그람이다. 이 프로그람은 기억장치에 항시적으로 존재하게 된다. 조작체계는 아래와 같은 두개의 매개물들과 대화한다.

- **응용프로그람**(문서편집쏘프트웨어와 같은)은 조작체계의 도움으로 실행된다.
- 사용자는 **지령언어해석기**(command language interpreter)를 리용하여 지령을 조작체계에 보낸다. 이것 역시 하나의 프로그람이며 지령들을 조작체계가 리해할수 있는 명령으로 넘긴다.

응용프로그람들은 파일의 열기와 닫기, 인쇄기접근, 테프에 쓰기 등과 같이 OS가 조종하는 봉사들을 자주 요구한다. 통보문을 구성할 때 무엇인가가 디스크자두를 이동시키고 자료를 디스크표면에 써넣는다. 응용프로그람은 그자체로가 아니라 바로 OS에 의거하여 그 일감을 수행한다. OS는 또한 봉사를 요구하는 사용자들을 대신하여 화면뒤에서 작업을 진행한다. 보다 명백히 말하여 조작체계는 다음과 같은 봉사들을 제공한다.

- 여러개의 프로그람이 동시에 실행되는 체계들에서 OS는 어느 응용프로그람을 즉시에 실행시키며 다음의 응용프로그람을 실행시키기전까지 얼마만한 시간을 할당하겠는가를 결정한다.
- OS는 여러 응용프로그람에 의해 공유되는 기억기를 관리한다. 그다음 응용프로그람이 실행될 차례가 되면 OS는 이전 응용프로그람의 자료를 디스크에 이동시켜 다시 적재될수 있는 상태를 유지하게 한다.
- OS는 하드디스크, 말단, 인쇄기 및 모뎀들과 같은 모든 하드웨어장치들에 대한 접근을 직접 조종한다.
- 운영과정에 오유와 맞다들리게 되면 OS는 명령을 보낸 매개물에 따라 오유내용을 그 응용프로그람과 사용자에게 통보한다.
- OS는 사용자들에게 등록부의 만들기와 파일들의 복사 및 삭제, 우편물의 전송이나 저장과 같은 기초봉사들을 실행하기 위한 도구들을 제공한다.

조작체계가 없이는 세계에서 가장 위력한 콤퓨터도 쓸모가 없게 되는것이다. 아무리 유능한 료리사라고 해도 조리도구를 일식으로 갖춘 부엌이 없이는 자기의 솜씨를 발휘할수 없는것과 마찬가지로 프로그람들이 제아무리 훌륭하다고 해도 조작체계의 도움이 없이는 기능을 전혀 수행할수 없는것이다.

과거에는 많은 조작체계들이 있었으며 매 하드웨어제작자에 따라 다르게 존재하였다. 그것들은 모두 무질서한 상태에서 사용되였으며 한 기계에서 개발된 프로그람은 다른 기계에서 실행시킬수 없게 되여 있었다. 두개의 서로 다른 기계들이 호상 대화해야 할 필요가 제기되면 제작자들은 소비자들에게 값 비싼 자기들의 하드웨어와 쏘프트웨어를 구입할것을 요구하였다. 그리하여 탁상용콤퓨터들에는 값이 눅고 놀랄만한 조작방법을 제공해 주는 DOS와 Windows들이 출현하게 되였다.

주해

조작체계는 프로그람들을 실행시키며 프로그람이 요구하는 모든 자원들을 조종한다. 또한 우편이나 파일, 프로쎄스들을 조작하기 위한 봉사도 제공한다.

1.2 UNIX조작체계

DOS와 Windows외에 UNIX라고 부르는 조작체계도 있다. UNIX조작체계는 DOS나 Windows보다 먼저 출현하였으며 오랜 기간 존재하면서 인터네트를 제공하였다. UNIX조작체계는 조작체계로서 갖추어야 할 모든것을 실지로 다 갖추고 있으며 지어 다른 조작체계가 가지고 있지 못한 여러가지 특징들도 가지고 있다. Windows에 대해 일정하게 경험해 본 초학자들은 UNIX를 Windows와 같은것으로 생각하면서 다만 외관상 비슷할뿐이라는 사실에 대하여 잊어 버리고 있다. 거대한 조작체계인 UNIX는 다른 체계들의 앞장에 서 있으며 절대적인 힘을 가지고 있다.

UNIX는 콤퓨터사회에 종전에는 알려 지지 않았던 심오하고 다양한 개념들을 많이 소개하였다. UNIX는 프로그람작성자들이 순수 자기들이 사용하기 위해 개발한 체계이므로 많은것들이 그들에게는 아주 명백하지만 우리모두에게는 그렇치 못하다. 그렇다고 하여 UNIX가 정복할수 없는 체계라는 의미는 아니다. 다만 UNIX는 그것을 인식하는데서 다른 형태의 노력이 요구될뿐이다. UNIX는 아주 불친절하다. UNIX는 흔히 사용자에게 정확한 동작을 하였는가 하지 못했는가에 대하여 말해 주지 않으며 사용자의 동작이 가져 오는 결과에 대해 경고해 주지 않는다.

UNIX는 또한 지령언어해석기를 가지고 있다. 사용자가 단어를 입력하게 되면 체계는 그것을 **지령**(command)으로 해석한다. 지령은 파일들을 목록화하거나 파일안에 있는 단어들을 계수한다. 하지만 UNIX의 힘은 이와 같은 단순지령들로부터 나오는것이 아니라 그것들을 서로 결합시키는 능력에서부터 나온다. 언어가 단어들을 결합시켜 의미심장한 뜻을 만들어 내듯이 UNIX지령들도 서로 문자렬을 이루어 복잡한 과제들을 형성한다. 사실 UNIX의 능력은 바로 사용자들의 상상력에 의하여 발휘된다.

만일 독자들이 일정한 지도를 받아 노력한다면 명백하게 그리고 알기 쉽게 펼쳐 지는 UNIX이야기들을 차차로 보게 될것이다. 본질적인 문제들에 주의를 집중하면서 설계자들의 생각과 대상들에 대하여 인식하기 위해 노력하여 보시오. 이따금 UNIX가 불친절한것처럼 보일수 있지만 사실 이것은 독자들에게 그자체가 안고 있는 수수께끼들을 정확히 해명해 주려는 목적으로부터 출발한 도전인것이다. 이 책에서 우리는 그러한 도전에 부닥쳐야 한다.

주해

UNIX체계는 고정된 봉사를 제공하지 않는다. 실질적으로 사용자는 현재 주어 진것으로부터 그때그때 요구되는 도구들을 자기의 상상력을 동원하여 창안해야 한다. 이것이 바로 UNIX를 매우 도전적이면서 자극적인것으로 되게 하는것이다.

1.3 콤퓨터에 대한 리해

Windows와는 달리 UNIX는 여러 사용자들이 동시에 사용할수 있다. 즉 한개의 디스크에 설치된 하나의 조작체계가 수백명의 사용자들에게 봉사할수 있다. 그러한 다중사용자체계에 접근하였다면 틀림없이 사용자는 한개의 말단(휴대용텔레비존처럼 보이는 감시기)과 한개의 건반앞에 앉아 있을것이다. 다른 사람들도 이와 류사한 말단들앞에서 작업하고 있을것이다. 나머지설비들은 대체로 접근이 제한된 다른 방에 배치되였을것이다. 이와 같은 상태에서 사용자는 자기의 등록자리(account)에 접근하여 작업을 수행한 다음 련결을 끊고 조용히 떠날수 있다.

하지만 **워크스테이션**(workstation)을 사용할 때에는 사정이 다르다. 이것은 고급한 도형들을 만들어 낼수 있는 콤퓨터이지만 한명의 사용자만이 리용할수 있다. 완전한 벙어리장치(dumb device)인 말단

과는 달리 워크스테이션은 자기의 고유한 CPU(중앙처리장치), 기억장치(RAM-자유호출기억기), 하드디스크, CD-ROM, 인쇄기를 가지고 있다. UNIX에 요구되는 모든것을 다 가지고 있는것으로 하여 워크스테이션은 UNIX를 실행시킬수 있다. 탁상용PC들도 자주 워크스테이션으로 간주된다.

워크스테이션이 비록 UNIX를 실행시키며 단독방식으로 사용될수 있지만 때때로 말단들처럼 보다 크고 강력한 콤퓨터와 련결된다. 그러한 설정을 요구하는데는 여러가지 원인들이 있다.

- 중앙콤퓨터는 원만히 관리되며 사용자는 자기의 가치 있는 파일들을 거기에 보관하고 정기적으로 여벌복사하려고 한다.
- 사용자는 자기의 워크스테이션에는 없고 중앙콤퓨터에 있는 강력한 프로그람을 사용하려 할수 있다.
- 들어 오고 나가는 사용자의 모든 우편이 외부세계 즉 인터네트와 유일하게 련결되여 있는 중앙기계에 의하여 조종된다.

이와 같은 기능들의 장점을 리용하자면 사용자의 워크스테이션이 망으로 중앙콤퓨터와 련결되여야 하는것이다. 사용자가 가지고 있는 기계가 망과 련결되여 있는가를 알기 위해서는 워크스테이션뒤에 중앙콤퓨터가 있는 방까지 설치된 선이 있는가를 알아 보면 된다. 매개의 워크스테이션은 자기의 보통방식을 버리고 간단한 벙어리말단처럼 작용하는 **말단모방**(terminal emulation)기능을 제공해 준다.

이렇게 되면 워크스테이션은 자기의 하드디스크나 CPU 또는 그 어떤 작업수행을 위한 기억장치를 사용하지 않게 되며 다만 말단모방쏘프트웨어가 요구하는 최소자원만을 제공하게 된다. 이 단계에서 사용자들이 새겨 두어야 할것이 하나 있다. 그것은 사용자가 중앙콤퓨터와의 련결을 위해 말단이나 혹은 말단모방기능을 사용할 때 모든 파일들은 사용자의 국부기계가 아니라 원격기계에 만들어 진다는것이다. 사용자가 실행시키는 프로그람은 또한 중앙콤퓨터의 기억장치와 CPU를 사용한다.

워크스테이션이나 PC를 가지고 있는 사용자는 그 콤퓨터의 시동과 체계정지 그리고 유지에 대한 직접적인 책임을 지게 된다. 만일 파일을 잃어 버리게 되면 여벌체계로부터 그것을 찾아 낼수 있다. 만일 작업이 정확히 수행되지 않게 되면 《수리공》을 부르기로 결심하기전에 이것들을 바로 잡기 위한 가능한 모든 수단을 다 리용해 보아야 한다.

주해

사용자의 워크스테이션이 현재 UNIX를 실행시킨다고 해도 사용자는 원격콤퓨터와 련결하여 거기에 있는 프로그람을 실행시켜야 할 필요가 있다. 이 경우에는 사용자의 워크스테이션이 단지 벙어리말단처럼 작용하게 된다. 사용자는 자기가 지금 원격기계에서 작업하고 있는지 아니면 자기의 국부기계에서 작업하고 있는지를 알려 주는 지시기가 있었으면 할것이다. 이에 대하여서도 이 책에서 취급하게 된다.

1.4 건반에 대한 리해

작업을 시작하기전에 사용자는 자기의 건반에 있는 많은 건들의 기능에 대하여 먼저 알아야 한다. 이 건들중 많은것들은 Windows체계에서 사용되지 않거나 다른 기능을 가진다. 우리는 여기서 PC건반에 대해 설명하게 되며 다른 체계들에서 볼수 있는 편향들에 대해서도 언급하게 된다.

왼쪽에 QWERTY가 배치되여 있는 건반(글자 Q, W, E, R, T, Y가 한선에 배렬되여 있다.)은 타자기와 류사하다. 수자건들은 이 행우에 배치되여 있다. 영어문자 및 수자건들의 배렬은 모든 체계에서 다 표준적이다. 타자치는 법을 알기만 하면 그와 류사한 건들과 마주 앉게 되므로 건반에 대한 공포 같은것이 생기지 않는다.

자모 및 수자건들과는 별도로 아래와 같은 기호들을 보게 될것이다

$ % * ( ) - = [ ] { } ; : ' " , . ? & / ! @

이 기호들은 모든 타자기들과 문서편집기들에서 사용되는것들로서 우리가 잘 알고 있는것이다. 이 기호들중 일부는 해당한 건을 [Shift]건과 함께 눌러 불러 낸다. 또한 건반은 독자들이 이전에는 보지 못했던 문자들도 일부 제공해 준다. 이에 대해서는 표 1-1에서 보여 주고 있다.

표 1-1. UNIX가 사용하는 특수한 문자들

부 호	이 름
`	역인용부호
~	물결표
#	파운드 또는 올림표
^	탈자기호 또는 고깔기호
_	밑선기호(-와는 다르다)
\	역사선
|	막대기호(관련결기호)
<	작기기호
>	크기기호

건반의 매개 건들은 UNIX에서 자기의 기능을 가지고 있다. 따라서 우리는 이 책을 더 보기에 앞서 먼저 매개 건들의 위치를 알 수 있게 되여야 한다. 많은 건들은 그것들을 사용하는 프로그람에 따라 여러가지 기능을 가진다. 실례로 |는 vi와 쉘에서 서로 다른 의미를 가지고 있다.

매개 자모와 수자, 기호들을 **문자**(character)라고 하며 문자는 사용자가 취급하는 정보의 최소단위를 나타낸다. 이 모든 문자들은 유일적으로 할당된 값들을 가지며 이것을 가리켜 **ASCII**(American Standard Code for Information Interchange)**값**이라고 부른다. 실례로 문자 G는 ASCII값 71을 가지며 감탄부호(!)는 33을 가진다. 많은 UNIX프로그람들도 ASCII값을 사용한다.

화면을 보면 깜박거리는것을 보게 되는데 이것을 가리켜 유표(cursor)라고 부른다. 문자의 건을 누르게 되면 유표가 있던 곳에 문자가 나타나고 유표는 오른쪽으로 이동하게 된다.

건반에는 타자기에서 찾아 볼수 없는 또 하나의 중요한 건이 하나 있는데 이것은 건반의 오른쪽에 있는 [Enter]건이다. 일부 기계들에는 [Return]이라고 되여 있다. 이 건은 행을 완료하는데 사용되며 그 의미에 대해서는 이 장의 뒤부분에 언급되여 있다.

[Enter]건의 바로 우에는 [Backspace]건이 있으며 흔히 →기호로 표시된다. 이 건은 오유를 퇴치하는데 사용된다(용어로서는 후진건이라고 부른다). 방금 기입한 문자를 지우려고 할 때 사용자는 이 건을 눌러야 한다. 그러면 유표가 왼쪽으로 이동해 가면서 문자를 삭제해 버린다.

다른 또 하나의 중요한 건은 건반의 왼쪽아래부분에 있는 [Ctrl](조종건이라고 함)이다. PC건반에는 이 건이 쌍으로 두개 있다. 즉 꼭 같은 건이 오른쪽에도 있다. 이 건은 절대로 혼자 사용되지 않으며 다른 건들과 결합되여 사용된다. 바로 이 결합이 이 책에서 맞다들게 되는 모든 **조종문자**(control character)들을 만들어 낸다. 실례로 화면이 흐르는것을 정지시키기 위해 [Ctrl-s]를 사용할 때에는 먼저 [Ctrl]건을 누르고 그 상태에서 s건을 눌러야 한다.

같은 선상에 [Alt]건도 있다(PC에는 이 건이 쌍으로 있다). 이 건 역시 결합되여 사용된다. 이 건은 emacs와 X Window체계를 사용할 때 필요하다. 많은 체계들은 이 건을 가지고 있지 않으며 그대신 [Meta]건을 가지고 있다. 만일 UNIX프로그람이 [Meta]건사용을 요구하는데 사용자건반에 그 건이 없을 때에는 보통 [Alt]건이 [Meta]건을 대신하게 된다.

왼쪽웃부분구석에는 [Esc]건(탈퇴건이라고 한다.)이 있다. 이 건은 본문편집기들인 vi와 emacs로 파일편집을 수행할 때 필요하다. [Esc]는 vi에서 매우 중요한 부분으로 된다. 마우스를 사용하지 않는 차림표(menu)에 기초한 프로그람에서 이 건은 흔히 사용자를 이전 차림표에 데려 간다.

오른쪽에는 [Delete]건이 있다. 일부 기계들에는 [Del]이라고 되여 있다. 이 건은 [Backspace]건이

없거나 있다고 해도 자기의 기능을 수행하지 못하는 체계들에서 그 건의 작용을 위해 사용된다.

[Esc]건이 있는 행에는 12개의 기능건들이 놓여 있으며 [F1], [F2], …, [F12]라고 표시되여 있다(낡은 체계들에는 [F10]까지 있다). 이 건들은 흔히 다르게 배치될수도 있다(이 건들을 건반의 왼쪽부분에서 볼 때가 있다). 이 건들은 같은 워크스테이션이나 PC에서(말단에서는 아니다.) 여러개의 **가상말단**(virtual terminal)들을 불러 낼 때 요구된다. 이 건들은 또한 vi와 같은 편집기들의 능력을 높이는데도 요구된다.

끝으로 모든 Windows사용자들이 잘 알고 있는 건들로 돌아 가보자. 유표조종건들(화살표가 있는 것)은 vi와 emacs에서의 항행(navigation)에 필요하다. 그 편집기들은 문서편집기에서와 꼭 같은 방법으로 [Home], [End], [Page Up], [Page Down]건들도 사용한다. 만일 사용자의 작업환경이 허용된다면 우아래로 된 화살표건반들을 사용하여 이전 지령들을 다시 호출할수 있다.

Linux

사용자는 자기의 워크스테이션이나 PC에서 [Alt]를 기능건들과 결합하여 사용하면 한대의 기계에서 여러개의 가상말단들을 가질수 있다. 이 가상말단들은 사용자가 한 기계에 여러번 가입하게 한다. 한개의 화면을 모든 가상말단들이 공유하므로 [Alt]건을 사용하여 이 가상말단들속으로 들어 가게 된다.

주해

특수문자를 나타내는 많은 건들과 지어는 [Ctrl]과 [Alt]건들까지도 체계들과 말단들에 따라 다르게 배치되여 있을수 있다. 일부 건반들은 [Meta]건도 가지고 있는데 PC건반에서는 [Alt]건이 그 건을 대치한다.

1.5 체계관리자

수백명의 사용자들에게 봉사하는 큰 체계에서는 누군가가 그 체계를 관리할 책임을 져야 한다. 이 사람이 바로 **체계관리자**(system administrator)이다. 체계관리자는 매우 중요한 사람으로서 전체적인 설치의 정확한 관리를 담당한다. 그는 사용자등록자리의 할당, 파일체계의 유지, 여벌복사(backup)의 수행, 디스크공간의 관리와 기타 중요한 몇가지 기능들을 수행한다. 문제가 생겼을 때 련계해야 할 사람이 바로 이 체계관리자이다.

UNIX는 보안기능을 가지고 있기때문에 콤퓨터체계에 **등록자리**(account)를 가지고 있는 사람들만이 사용할수 있다. 이 등록자리들의 목록은 콤퓨터에 제각기 따로 보존된다. 관리자는 자기가 사용하는 특정한 사용자등록자리를 따로 가지고 있는데 이것을 가리켜 **뿌리**(root)라고 한다. 뿌리는 거의나 절대적인 권한을 가지고 있으며 일부 프로그람들은 이 등록자리에 의해서만 실행된다. 실례로 사용자등록자리 그자체를 만드는 프로그람을 들수 있다.

관리자는 사용자가 사용할 이름으로 등록자리를 개설한다. 이 이름은 보통 의미가 있는 문자렬로서 **사용자식별자**(user-id:간단히 사용자ID라고 함) 혹은 **사용자이름**(username)으로 불리운다. 또한 사용자는 체계가 문의할 때 입력해야 할 비밀코드를 제공 받게 된다. 이 문자렬을 **통과암호**(password)라고 부른다. 사용자ID와는 달리 통과암호는 의미가 없는 문자렬로 되여야 한다. 효과적인 사용자ID와 통과암호로 체계에 먼저 가입하지 않고서는 사용자가 그 어떤 말단에 앉아서 작업을 시작할수 없다.

1.6 체계가입 및 탈퇴

백문이 불여일견이라는 말도 있는것처럼 이제는 천천히 일에 착수하면서 UNIX체계에 직접 뛰여 들어 UNIX가 실지로 어떤가를 체험하여 보자. 이 체계와 직접 대화해 보는것이 때로는 교과서들(이 책도 포함)에서의 설명보다 결승점에 더 빨리 도달하게 해줄수도 있다.

1.6.1 체계가입

만일 사용자가 큰 체계에서 작업하면서 말단을 제공 받았다면 관리자에게 사용자의 ID(식별자)와 통과암호를 요구해야 한다. 사용자가 자기의 워크스테이션에서 UNIX를 시험하는 경우 사용자는 자체로 할 자신이 없다면 일정한 UNIX경험을 가진 사람들에게 등록자리를 설치해 줄것과 이 두개의 파라메터를 보장해 줄것을 요구할수 있다.

사용자가 자기의 사용자ID로서 romeo라는 이름을 가지고 있다면 말단에서 아래와 같은 **프롬프트**(prompt)에 그 이름을 입력하면 된다.

```
login:
```

프롬프트문자렬의 단어 login:에는 여러 접두사들이 있을수 있으며 때로는 거기에 기계이름이 있는것도 볼수 있다(UNIX에서는 모든 기계들이 이름을 가지고 있다). 일부 체계들에서는 그대신 Username:을 보게 된다. login:프롬프트는 누구인가가 그 말단에 **가입**(log in)할수 있다는것(즉 기계상의 자기 등록자리에 련결하는것)을 가리킨다. 이 통보문은 또한 이전 사용자가 **탈퇴**(log out)하였다는것(즉 자기 일을 끝내고 련결을 끊었다는것)을 가리키기도 한다.

현재 romeo라는 이름의 등록자리를 가진 상태에서 이 문자렬을 프롬프트에 입력하고 그뒤에 [Enter]건을 눌러 보자.

login: **romeo**[Enter]
password:

체계는 지금 관리자가 사용자에게 준 비밀코드인 통과암호를 기입할것을 요구하고 있다. 이 코드는 사용자를 제외한 다른 사람들에게는 로출되지 않는다(관리자도 이것을 알 필요조차 없다). 이것은 마치 자동계산기에서 돈을 찾기전에 건입력해야 하는 개인확인번호와 같은것이다. 비밀코드를 입력하고 [Enter]건을 누르자.

login: **romeo**
password: *********[Enter]　　　　　　표시되지 않는다

사용자는 자기가 기입하는 통과암호가 화면에 나타나지 않는것을 보고 놀랄수 있다. 이것은 체계에 구축된 또 하나의 보안기능이다. 누구든지 사용자곁에 서 있던 사람은 사용자가 무엇을 입력했는가에 대해 볼수 없게 되여 있다(그렇지 않으면 사용자의 손가락놀림을 주시하느라 신경을 써야 할것이다).

만일 타자를 치다가 실수하는 경우에는 체계가 허용되는데 따라 [Backspase]건으로 지울수 있다. 사용자는 또한 [Enter]건을 한번 또는 두번 눌러서 체계가입프롬프트가 다시 화면에 나타나게 할수도 있다. 사용자가 입력한것을 체계가 읽어 들이도록 하자면 사용자의 응답을 [Enter]로 완료해야 한다. 만일 사용자가 사용자ID나 통과암호를 틀리게 입력하면 체계는 다음의 통보문을 내보낸다.

```
login incorrect
login:
```

이것은 또 하나의 보안준위이다. 사용자는 무엇이 잘못되였는지 즉 가입이름인지 아니면 통과암호인지 모른다. 통보문 login incorrect는 사실상 완전히 믿을수 없다. 대다수의 경우 장본인은 통과암호이다. 이 파라메터들을 모두 정확히 해놓게 되면 다음과 같은것을 보게 된다.

```
Last login: Thu Sep 16 22:11:17 on tty2
$ _
```

여기서 체계는 과거에 사용자가 마지막으로 가입한 시간을 보여 주는 통보문을 나타내고 있다. 사용자의 체계는 다른 통보문들도 보여 줄수 있다. 최종적으로 사용자는 프롬프트문자렬(여기서는 단지 $)곁에서 유표(_문자)가 계속 깜박거리는것을 보게 될것이다.

사용자가 다음행동을 생각하고 있는 동안에도 사용자의 말단에서는 프로그람이 계속 실행되고 있다. 이 프로그람은 프롬프트를 만들고 사용자의 건반으로부터 모든 입력을 접수한다. 이것을 **쉘**(shell)이라고 하며 UNIX체계는 여러가지 쉘을 제공하여 사용자가 선택하도록 한다. 쉘은 지령언어해석기이다. 즉 조작체계로 대화하는 두개의 매개물들중 하나이다.

UNIX체계들은 대체로 $를 기정프롬프트로 특징 지어 준다. 하지만 학계에서는 $보다 %가 더 일반적이다. 이번에는 유표를 보여 주지 않는다.

```
%                  C쉘은 이것을 사용한다
```

사용자가 쉘을 사용하는데 따라 프롬프트들은 전용화되여 사용자ID와 기계이름, 현재의 날자와 시간을 보여 줄수 있다. 더우기 체계관리자는 대체로 #를 자기의 프롬프트로 사용한다. 아래에 또 다른것이 있다.

```
[/home/romeo]                      쉘은 등록부를 보여 줄수 있다
```

이것은 사용자의 현재등록부를 보여 주고 있다. 현재등록부는 AUTOEXEC.BAT에서 PROMPT [pg]문를 사용하여 프롬프트문자렬을 전용화하는 대부분의 Windows사용자들에게 잘 알려 진 개념이다. 이 책에서는 대체로 $를 프롬프트로 한다.

주해

쉘은 사용자가 가입하여 있는 전 기간 계속 실행되는 프로그람이다. 사용자가 프롬프트에서 입력하는 모든것은 사실 이 프로그람에 입력된다. 관리자는 사용자의 등록자리를 개설할 때 사용자의 쉘을 설정해 주지만 때때로 사용자자신이 그것을 변경시킬수도 있다.

Linux

Linux체계들은 거의나 프롬프트로서 $나 %를 보여 주지 않는다. 이 체계들은 기계(saturn), 사용자의 이름(romeo), 현재등록부(/home/romeo)를 보여 주는 아래와 같은 매우 유익한 프롬프트들로 미리 구성되여 있다.

```
romeo@saturn:/home/romeo >
```

1.6.2 체계탈퇴

체계에 유용한 일부 UNIX지령들에 대한 시험을 계속하기에 앞서 이 체계에서 탈퇴하는 법(사용자자신을 체계에서 꺼내는 법)을 먼저 알아야 한다. 여기에 대한 기술 역시 사용자가 리용하는 쉘에 의존하므로 이 모든것(기본적으로 3가지)들을 차례로 해보아야 한다. 먼저 [Ctrl-d]를 해보도록 하자.

```
$ [Ctrl-d]             [Ctrl]을 계속 누르고 있는 상태에서 d를 누른다
login:
```

[Ctrl-d]는 [Ctrl]건과 문자 d를 함께 눌러서 얻는다. login:통보문은 대화기간이 완료되였으며 다음사용자가 말단을 리용할수 있다는것을 알려 준다. login:프롬프트가 보이지 않고 아래와 같은것이 보일수 있다.

```
Use "logout" to log out.
```

이것은 [Ctrl-d]가 작용하지 않으며 사용자가 대신 logout지령을 사용할수 있다는것을 의미한다. 이것은 일상적으로 %를 프롬프트로 사용하는 등록자리들에서 생긴다.

```
% logout[Enter]                    여기서 %는 프롬프트이다
login:
```

만일 logout지령이 동작하지 않으면 exit지령이 그 일을 수행한다. 대체로 이 지령은 대화기간을 완료하는 지령이다.

참고 3개의 지령들인 [Ctrl-d], logout, exit가운데서 어느 하나를 사용하여 체계탈퇴를 할수 있다. logout지령은 일반적으로 %프롬프트를 만드는 C쉘에서 동작한다. [Ctrl-d]와 exit는 다른 쉘들이 사용한다.

1.7 일부 지령들의 시험사용

두개의 프롬프트에 사용자이름(사용자ID)과 통과암호를 주고 다시한번 체계에 가입하여 보자. 가입을 성과적으로 하게 되면 사용자는 UNIX가 제공하는 거의 모든 훌룡한것들에 자유롭게 접근할수 있게 된다. 사용자가 건으로 단어를 입력하면 체계는 그것을 지령으로 해석한다. 사실 사용자는 기계에 그 어떤 작업을 명령하고 있는 상태에 있다. 모든 지령들이 다 접수되는것이 아니라 체계가 사전에 결정한 목록에 있는것만 인정한다. 우리는 이제 이러한 몇개의 지령들에 대하여 파악하여 보자.

1.7.1 사용자통과암호의 변경(passwd)

우리는 체계가입시에 입력하는 통과암호를 변경시키는것으로써 체계와의 의미 있는 첫 대화를 시작하게 된다. 그것을 위해 요구되는 지령은 passwd이다.

```
$ passwd[Enter]
Old Password : ********[Enter]
New Password : ********[Enter]
Reenter Password : ********[Enter]
$ _              프롬프트가 귀환되며 지령은 완성된다
```

단어 passwd를 기입하고 뒤이어 [Enter]를 누르게 되면 사용자의 응답을 3번 요구하게 된다. 먼저 사용자가 이 프롬프트에서 낡은 통과암호를 입력한다. 다음 사용자가 입력한 통과암호가 유효한가를 검사해 보고 유효한 경우에는 새 통과암호를 문의한다. 체계에 적용할수 있는 통과암호설정규칙들을 리용하여 통과암호를 입력한다. 마지막으로 체계는 사용자가 새 통과암호를 다시 입력할것을 요구한다.

예상한대로 사용자가 입력한 문자들이 말단에 현시되지 않는다. 때문에 사용자는 주의해서 천천히 입력하여야 한다. 새 통과암호를 위한 두개의 입력이 서로 맞지 않으면 체계는 오유통보문을 내보내며 사용자는 passwd지령을 다시 호출하지 않으면 안된다. 지령실행이 완성되면 $프롬프트가 귀환된다.

통과암호만들기규칙들은 사용하는 UNIX종류와 거기에 설정된 보안준위에 의존한다. 문자렬에는 때때로 최소길이가 정해 져 있다. 많은 체계들이 수자와 글자들의 결합사용을 주장하고 있다. 일반적으로 글자 및 수자들의 대문자와 소문자결합은 다른 사람들이 통과암호의 의미를 알수 없게 만든다. 이것은 권한 없는 다른 사용자가 자기의 령역에 순수 짐작으로 들어 오게 되는것을 방지한다. 독자들은 나쁜 목적을 품은 사람들이 어떻게 해서든지 다른 사람의 통과암호가 무엇인가를 알아 내는 경우 초래될 무서운 결과에 대해 나중에 알게 될것이다.

다음번에 가입할 때에는 체계에 들어 가자면 반드시 새 통과암호를 기입해야 한다는데 대해 주의를 돌려야 한다. 사용자는 통과암호를 바꾸기전까지는 통과암호이름을 기억하고 있어야 한다. 만일 잊어 버렸으면 즉시 체계관리자에게 통보해야 한다.

passwd는 UNIX체계에서 쓰이는 수백개의 지령들중의 하나이다. 우리는 뒤장에서 passwd지령에 대해 구체적으로 취급하게 된다.

주해

$프롬프트가 귀환되면 이것은 이전 지령과 관련된 모든 지령이 수행되였으며 다음지령을 접수할 준비가 되였다는것을 의미한다.

주의

때때로 체계는 사용자의 통과암호를 변경하는것이 너무 이르다고 간주할수도 있다. 그때는 지정된 기간동안 기다린후에 통과암호를 바꾸어야 한다. 체계관리자에게 이에 대한 규정을 늦추어 줄것을 요구할수도 있다.

1.7.2 사용자들을 알아보기(who)

UNIX는 많은 사용자들이 사용하는 체계이므로 사용자는 현재체계를 사용하고 있는 사람들에 대해 알고 싶을수도 있다. 이 경우에 who지령을 사용한다.

```
$ WHO[ENTER]
ROMEO    TTY01       MAY 08 11:11
HENRY    TTY02       MAY 08 14:18
STEVE    TTY03       MAY 08 12:01
```

현재 여기에는 3명의 사용자들인 romeo, henry, steve가 있다. 이것들은 그들이 체계에 가입하는데 사용한 사용자ID들이다. 말단이름들과 가입한 날자, 시간도 보여 주고 있다.

사용자는 또한 자기의 ID도 알아야 한다. 사용자는 romeo의 이름으로 가입하였기때문에 체계는 이 이름으로 사용자를 주소화하며 사용자가 진행하는 작업들은 모두 romeo와 련결되게 된다. 때때로 사용자는 자기 이름을 잊어 버리게 되는데 특히 체계안에 여러개의 등록자리를 가지고 있는 경우에는 더욱 그렇다. 자기의 ID를 알자면 같은 who지령을 사용하되 이번에는 두개의 단어 am과 i를 추가적으로 사용한다.

```
$ who am i[Enter]
romeo    tty01       May 08 11:11
```

그밖에 또 어디에 사용자이름이 요구되는가? 파일을 만들면 체계는 romeo를 그 파일의 **소유자**(owner)로 만들것이다. 다른 사용자에게 우편을 보내게 되면 체계는 수신자에게 이 우편이 romeo로부터 온것이라고 통지해 준다. 기계는 romeo가 마치 사용자의 진짜이름인것처럼 이 이름으로 사용자를 식

별한다. 이것은 체계관리자가 사용자에게 사용자ID를 할당할 때에도 마찬가지이다.

who와 함께 사용되는 추가적인 단어는 **인수**(argument)들이며 대부분의 UNIX지령들은 몇개의 인수들과 함께 사용된다. 인수들은 지령 그자체의 동작을 변경시킨다. 이 책에서는 기본적인 UNIX지령들이 가지고 있는 중요한 인수들에 대하여 설명하고 있다.

who출력에는 여러가지 렬들이 무엇을 의미하는가에 대하여 알려 주기 위한 머리부가 들어 있지 않는다. 이것은 이 체계의 아주 중요한 특징으로서 그의 친절치 못한 측면을 보여 주는 한 실례이다.

주해

who와 함께 사용되는 추가적인 단어들을 인수라고 한다. 앞에서 본 am과 i도 인수이며 후에 다른 인수들에 대해서도 배우게 된다.

1.7.3 말단이름알아보기(tty)

사용자자신이 누구인가를 안 다음에는 사용자의 말단이름도 알아야 한다. 이것은 tty01로 보이지만 사실 UNIX는 사용자에게 이것을 알려 주는 지령을 따로 가지고 있다. 그것이 바로 tty지령이다.

```
$ tty[Enter]
/dev/tty01
```

여기서 보다싶이 tty01이라는 이 이름에는 추가적인 구성요소가 존재한다. UNIX의 유명한 특징들중의 하나가 모든 장치들을 **파일**(file)로 간주하는것이며 tty01도 체계의 한개 파일로 된다. 현재는 그것이 dev라고 하는 폴더에 위치하고 있다고 가정하자.

주해

모든 사용자들이 이름을 가지고 있는것처럼 모든 말단들과 디스크들, 인쇄기들도 이름을 가진다. 후에 우리는 이 모든 이름들이 체계안에서 파일로 나타나는것을 보게 될것이다. 모든 지령들도 역시 파일이다.

1.7.4 사용자의 쉘과 말단형에 대하여 알아보기

$와 다른 프롬프트문자렬을 가지고 작업하는것도 가능하다. 이것은 보통 사용자가 사용하는 쉘에 의해 결정되게 된다. echo지령을 $SHELL인수와 함께 사용하면 사용자쉘의 이름을 현시하게 된다.

```
$ echo $SHELL[Enter]                    $로 평가된다
/bin/ksh              Korn쉘
```

이것은 bin폴더에 배치된 파일 ksh에 의해 표현되는 Korn쉘이다. 모든 지령들이 다 파일인것처럼 쉘 역시 파일이다. $SHELL(SHELL이라고도 한다.)은 UNIX환경에서 사용되는 **변수**(variable)이다. 프로그람작성변수들과 마찬가지로 쉘변수들 역시 자기들에게 해당한 값을 가진다. echo지령은 임의의 변수값을 평가하는데 사용한다.

SHELL을 비롯하여 수많은 변수들이 있다. 그 변수들중 하나가 바로 사용자가 사용하는 말단형을 알려 주는 TERM이다.

```
$ echo $TERM[Enter]
ansi
```

이것은 사용자의 말단은 이름이 /dev/tty01이지만 형은 ansi라는것을 말해 주고 있다. 서로 다른 말단들은 서로 다른 특성을 가지며 일부 지령들(vi편집기와 같은)은 사용자의 말단형에 대하여 알 필요가

있다. 만일 사용자의 말단형이 부정확하게 설정되면 여러가지 문제들에 부닥치게 되며 특히 원격기계에서 작업할 때는 더욱 그러하다.

1.7.5 체계환경알아보기(set)

일반적으로 UNIX지령들은 소문자로 되여 있지만 모든 체계변수들은 대문자로 되여 있다. SHELL과 TERM은 오직 대문자로만 정의된다. shell이라고 쓰면 다른 변수를 사용하는것으로 될것이다(사용하고 있는 쉘에 따라 정의되지 않을수도 있다).

모든 체계변수들의 목록을 현시하기 위해서는 set지령을 사용해야 한다. 이 목록은 길어 져서 화면밖으로 넘쳐 날수 있다. 아래에 중요한 변수들을 요약한 목록을 준다.

```
$ set[Enter]
CDPATH=.:..:/home/romeo
HOME=/home/romeo
HOSTNAME=saturn
MAIL=/var/mail/romeo
MAILCHECK=60
PATH=/home/romeo/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:.
PS1='$ '
SHELL=/bin/ksh
```

여기서 매 변수들은 체계기능의 중요한 부분을 조종한다. 실례로 사용자는 PS1값을 변경시켜 프롬프트가 어떻게 보이도록 하겠는가에 대해 결정한다.

주해

set지령의 출력은 사용자가 사용하는 쉘에 따라 변경된다. 만일 echo $SHELL이 /bin/csh를 나타내게 되면 출력은 약간 달라 진다(=가 있는 곳에 공간이 생긴다). /bin/csh는 C쉘을 나타내는데 흔히 다른 쉘들과는 다르게 동작한다. 이 책에서는 전 기간에 걸쳐 쉘의 동작상 차이점들에 대하여 특별히 언급하고 있다. 만일 지령이 이 책에서 설명한대로 출력을 만들지 못한다면 그것은 대체로 쉘때문이다.

이 장의 마지막에 또 다른 대화기간을 취급하므로 현재는 다음사용자에게 길을 내주기로 하자. 이 작업기술에 대해 지금 당장 알아야 한다. 일반적으로 exit는 대화를 완료한다.

```
$ exit[Enter]
login:
```

참고

작업이 끝난 다음에는 언제나 콤퓨터를 탈퇴시켰는가를 확인하여야 한다. 그렇게 하지 않으면 다른 사용자가 쉽게 당신의 파일들을 삭제할수 있다.

1.8 두가지 중요한 고찰

체계가입후 사용자는 체계와 첫 대화를 하였다. 다음대화로 넘어 가기전에 이전 대화의 결속으로서 두가지 중요한 문제점을 보기로 하자. 독자들자신이 이 문제들에 대해 관찰하지 못했다면 다음단락들에서 그에 대해 보여 준다.

1.8.1 UNIX지령들은 소문자로 되여 있다

UNIX에서 문자의 크기는 중요하다. UNIX지령들은 일반적으로 소문자로 되여 있으며 만일 사용자가 대문자를 사용하게 되면 체계는 그것을 거절한다. passwd지령대신에 PASSWD를 사용하여 보자.

```
$ PASSWD[Enter]
/bin/ksh: PASSWD: command not found
```

체계에는 PASSWD지령이 없다. 하지만 이것은 사용자가 지령을 만들 때 대문자를 사용하는것을 막지는 않는다. 소문자는 규정이라기보다는 관습이다.

참고

UNIX체계를 사용할 때 [Capslock]건을 누르지 않았는가를 확인하시오. 이 건은 vi나 emacs와 같은 편집기들에 대문자로 된 본문을 삽입할 때를 제외하고는 어디서나 말썽을 일으킨다.

1.8.2 [Enter]건

사용자는 입력한 매행의 마감에 [Enter]건을 누른다. 말단에서 타자치는 본문은 이 건을 누르기전까지는 체계로부터 계속 숨겨 져 있다. $나 %프롬프트에 입력하는 모든것은 원칙적으로 뒤따라 [Enter]를 눌러 주어야 한다.

인간정보체계에는《[Enter]건》이 없기때문에 첫 사용자들은 이 점에 대해 놓치군 한다. 사람들은 말과 본문을 련속적으로 말하고 읽는 방법으로 입력한다. 여기서 [Enter]는 사진기의 샤타와 비슷하다. 샤타를 누르기전에는 필림에 아무것도 기입되지 않는 법이다. 이 건의 필요성은 명백하므로 앞으로 애매한 경우를 제외하고는 [Enter]건에 대한 상세한 취급을 피하려고 한다.

1.9 제대로 안되는 경우

말단과 건반들은 통일적인 동작형태를 가지고 있지 않다. 말단설정은 건반작용에 직접적인 영향을 미치므로 TERM변수값을 자주 검열해 보아야 한다. 이 변수의 중요성에 대해 론하는것은 너무 이르지만 독자들은 적어도 일부 공통적인 함정에서 빠져 나갈수 있게는 되여야 한다. 사용자는 자기가 생각했던것과는 완전히 다르게 동작할 때 어느 건을 누르겠는가에 대해 알고 있어야 한다.

1.9.1 Backspace가 동작하지 않는다

지령행에서 몇개의 오유를 발견하고 [Backspace]건을 눌렀는데 본문이 지워 지지 않았다. 실례로 사용자가 passwd대신에 password라고 잘못 입력했다면 마지막 3개 문자들(ord)을 지우고 d를 기입해야 한다. [Backspace]건을 3번 누르면 다음과 같은것을 보게 될수도 있다.

```
$ password^H^H^H
```

여기서 [Backspace]가 동작하지 않기때문에 그 건을 누를 때마다 ^H부호를 보게 된다. 이와 같은 현상은 사용자의 국부기계와 차이나는 말단설정을 가진 원격기계에 가입할 때 종종 보게 된다. 사실 싼회사의 기계들은 [Backspace]건을 전혀 사용하지 않는다. 이런 경우에는 아래의 두개 건을 시도해 보아야 한다.

```
[Ctrl-h]
```

[Delete] 또는 [Del]

이것들중 하나가 사용자에게 맞을수 있다. 해당한 건을 눌러서 지우기(erase)문자를 만들어 낸다. 이 문자를 만드는 건을 변경시킬수도 있는데 이와 관련한 전용화기술에 대해서는 뒤장에서 론의하게 된다.

1.9.2 지령을 중단시켜야 한다

비록 이것이 이 장에서 특징 지어 주고 있는 지령들에서는 일어 나지 않는 현상이기는 하지만 어쨌든 일반적인 문제거리이다. 다음대화기간에 사용자는 파일이름을 표현하는 단어와 함께 cat지령을 사용하게 된다. 이제 cat를 입력하고 곧 [Enter]건을 우연히 눌렀다고 가정하자.

$ **cat**[Enter]

여기서는 아무런 동작도 일어 나지 않는다. 지령은 단지 사용자로부터 그 어떤 입력을 기다리고 있다. 입력하는 경우에도 사용자는 자기의 입력을 어떻게 완료하겠는가에 대해 알아야 한다. 사용자입력을 기다리는 지령을 위해 [Ctrl-d]를 입력하여 프롬프트가 돌아 오게 한다.

$ **cat**[Enter]
[Ctrl-d]　　　　　파일의 끝을 나타낸다
$ _

이 건은 파일끝표식을 나타내며 eof문자로서 표현된다. 이 문자는 변경될수도 있다.

때때로 프로그람은 1시간동안 계속 실행하면서 사용자입력을 기다리지 않는 경우도 있다. 이때 [Ctrl-d]는 동작하지 않으며 사용자는 두개의 건들중에서 어느 하나를 사용하여 프로그람을 중단시켜야 할것이다.

[Ctrl-c]
[Delete]　　　　　프로그람을 중지하기 위하여

프로그람을 비정상적으로 완료시키기 위해 대부분의 UNIX체계들은 [Ctrl-c]를 사용하지만 일부 체계들은 [Delete]건을 사용하는것도 있다. 해당 건은 **새치기문자**(interrupt character)를 만든다.

1.9.3 행의 제거

만일 행에 오유가 많은 경우에는 그것을 실행시키지 말고 차라리 행전체를 제거하여 버리는것이 더 낫다. 이 경우에는 다음의 건을 사용할수 있다.

[Ctrl-u]　　　　　행을 제거한다

이것을 가리켜 **행제거문자**(line kill character)라고 한다. 이것은 프로그람을 제거하는 새치기문자와는 명백히 다르다. 행제거문자는 지령이 실행되기전에 지령행에 있는 모든것을 지워 버린다.

1.9.4 기타 문제들

우의것들외에도 사용자는 다른 문제들에 부닥칠수 있다. 그에 대해 아래에 간단히 서술한다.

- 만일 일감이 중요해서 중단시키지 말아야 한다면 [Ctrl-z]로 정지만 시킬수 있다. 그다음에는 보다 더 중요한 일감을 실행시키고 fg지령으로 다시 정지된 일감을 시작할수 있다.
- 만일 지령에 의해 나타난 화면이 너무 빨리 흘러 가기때문에 말단을 볼수 없다면 [Ctrl-s]를 눌러서 출력을 림시 멈춰 세울수 있다. [Ctrl-g]를 눌러서 흐르기(scrolling)를 다시 시작한다.

- [Enter]건을 사용하여 지령행을 완성한다. 만일 이 건이 동작하지 않으면 [Ctrl-j] 또는 [Ctrl-m]을 사용할수 있다.
- 엉뚱하게 동작하는 말단(특히 전화면편집기로 파일을 편집할 때)에 대해서는 stty지령을 사용하여 자기 상태로 돌아 오게 한다. 이와 같은 환경에서는 [Enter]건이 작용하지 않으므로 [Ctrl-j]나 [Ctrl-m]을 사용한다.
- 만일 사용자가 자체의 프롬프트를 가지는 어떤 프로그람으로부터 벗어 날수 없게 되면 q, quit, exit, [Ctrl-d]를 사용하여 프로그람을 정지시킬수 있다.

이 건반기능들에 대해서는 표 1-2에서 개괄하였다. 하지만 이 기능들은 체계에 의존하며 일부 건들은 여기에 언급한것과 다르게 동작할수도 있다. 만일 문제가 생기게 되면 체계관리자에게 도움을 청할수도 있다.

표 1-2. 건반지령들

건조작 또는 지령	기 능
[Ctrl-j]	[Enter]와 같이 리용된다
[Ctrl-m]	우와 같다
[Ctrl-h]	본문을 지운다([Backspace]건이 동작하지 않는 경우)
[Ctrl-c]	지령을 중단한다(이것이 실패하면 [Delete]나 [Del]을 사용한다)
[Ctrl-u]	실행함이 없이 그 지령행을 제거한다
[Ctrl-\]	실행중의 지령을 제거하며 그 프로그람의 기억기영상을 포함하는 주기억파일(core file)을 만든다
[Ctrl-s]	화면출력의 흘리기를 정지한다
[Ctrl-q]	화면출력의 흘리기를 재개한다
[Ctrl-d]	가입대화기간이나 건반으로부터 입력을 기다리는 프로그람을 완료한다
exit	가입대화기간을 완료한다(항상 동작한다)
logout	C쉘에서 가입대화기간을 완료한다
[Ctrl-z]	프로쎄스를 중지하고 쉘프롬프트를 돌려 준다(fg를 사용하여 일감을 재개한다)
stty sane	말단을 보통상태로 회복한다(UNIX지령)

주해

쉘이 일감조종을 지원할 때에만 사용자는 [Ctrl-z]로 일감을 중지할수 있다.

1.10 파일 및 등록부에 대한 작업

앞에서 진행한 대화기간에 리용한 지령들은 파일과 **등록부**(directory:종전에는 폴더라고 하였다.)들에 귀속된다. who am i(tty0)에 의해서 현시된 말단이름은 하나의 파일이다. tty지령은 파일 tty01의 완전한 경로이름으로서 /dev/tty01을 보여 준다. set지령출력에서 보게 되는 많은 변수들도 파일과 등록부들을 가리킨다.

사용자이름과 통과암호를 사용하여 체계에 다시 가입하여 보자. 사용자의 통과암호를 변경한 다음에는 새 통과암호를 넣어야 한다는것을 명심해야 한다. 일단 $프롬프트에 들어 오게 되면 사용자는 다음의 지령들을 리용할 준비가 된것으로 된다. 이제 우리는 파일과 등록부들을 만드는 법과 한 등록부로부터 다른 등록부에로 이동하는 법을 배우게 된다.

1.10.1 등록부만들기(mkdir)

UNIX는 등록부라고 부르는 서로 다른 폴더들에 자기의 모든 파일들을 구성한다. 사용자는 mkdir라고 부르는 지령으로 등록부를 만들수 있다.

```
$ mkdir docs
$ _              cs등록부가 만들어 졌다
```

출력은 없다. 모든 UNIX지령들이 출력을 만들지는 않으며 일부 지령들은 일감을 수행한 다음 곧바로 프롬프트로 돌아 온다. 새로운 등록부가 만들어 졌는지 확인하여 보자.

1.10.2 파일 및 등록부들의 목록얻기(ls)

ls지령(list를 반영한 략어지령)은 모든 파일과 등록부들을 현시한다. 지금까지 이전의 대화기간에서 론의된 지령들만 실행하였다면 사용자가 하나의 등록부만을 만들었을뿐 파일을 한개도 만들지 않았으므로 ls는 많은것을 현시하지는 않을것이다.

```
$ ls
docs
```

ls는 **현재등록부**(current directory)에서 유일한 파일로서 docs를 현시한다. 그러면 이것은 파일인가, 등록부인가, 혹은 둘 다인가? UNIX는 파일이라는 용어를 매우 자유롭게 취급하며 파일정의에 등록부들도 포함시킨다. 제6장에서 파일과 등록부들에 대한 많은 내용들을 설명하게 된다.

1.10.3 등록부구조의 조종(pwd와 cd)

mkdir외에도 UNIX는 등록부들을 조종하는 특수한 지령묶음을 가지고 있다. UNIX는 등록부들을 마치 사용자가 들어 갈수 있는것과 같은 공간의 차원에서 취급한다. cd지령은 사용자의 현재등록부를 변경하고 사용자가 다른 등록부에 놓이게 하는데 사용된다. 이러한 작업을 하기전에 먼저 사용자의 현재등록부가 무엇인가부터 알아야 한다. 이것은 pwd지령이 수행하는 일감이다.

```
$ pwd
/home/romeo              현재등록부
```

사용자는 자기의 사용자ID에 따라 이름 지어 진 등록부를 가진다. 사실 이 등록부는 사용자등록자리가 개설될 때 만들어 진것이다. 이것은 사용자가 가입한 다음 위치하게 되는 등록부 즉 **홈등록부**(home directory)이다. 이제는 cd지령을 사용하여 만들어 진 docs등록부로 이동하여 보자.

```
$ cd docs
$ _
```

여기서는 아무런 통보가 없는것으로써 지령이 실수없이 실행되였다는것을 의미하고 있다. 사용자는 docs등록부에 있게 된다. 그것을 확인하기 위하여 pwd지령을 다시 사용하여 보자.

```
$ pwd
/home/romeo/docs
```

이것은 등록부 docs를 마지막구성요소로서 보여 주는 **경로이름**(pathname)이다. docs는 romeo등록부아래에 놓이며 romeo는 home등록부아래에 놓인다. 최상위등록부는 경로이름의 첫 사선(/)이다(이것

을 뿌리등록부라고 부른다).

1.10.4 파일의 만들기와 보기(echo와 cat)

사용자는 현재 docs등록부에 있다. 여기서 일부 파일들을 만들어 보자. UNIX는 2개의 훌륭한 편집기들(vi와 emacs)을 가지고 있는데 후에 이것들을 사용하여 파일들을 만들고 편집하는 방법을 보게 될것이다. 하지만 여기서는 보다 간단한 방법을 사용하게 된다. 즉 echo지령을 사용하여 This is the first message라는 단어들을 파일 note1안에 배치하여 보자.

```
$ echo This is the first message > note1
$ _                    5개 단어를 가진 파일을 만든다
```

종전의 echo지령(echo $TERM)은 말단에 그 무엇인가를 현시하였지만 이번에는 그렇치 않다. 종전의 지령은 >부호를 전혀 사용하지 않았으며 이것이 차이점의 근원으로 되였다. >는 파일 notel에 echo지령출력을 보관하였다.

이제는 또 하나의 파일을 만들고 그 파일을 note2라고 부르기로 하자.

```
$ echo This is the second message > note2
$ _
```

우리는 지금 docs등록부에 2개의 파일들을 만들었다. 우리는 등록부 docs를 변경시키지 않았으므로 ls는 이 2개의 파일이름을 보여 줄것이다. ls지령을 다시 실행하여 보자.

```
$ ls
note1    note                    두개의 파일을 보여 준다
```

이것들은 물론 예상했던 그대로이다. 이 파일들안에 무엇이 있는가를 보기 위해 **만능파일열람기**(universal file viewer)인 cat지령을 사용한다. 이 지령은 앞에서(1.9.2)도 사용하였으며 그때에는 [Enter]를 즉시 눌렀었다. 이번에는 지령후에 파일이름 notel을 입력하여 보자.

```
$ cat notel
This is the first message                  파일내용을 보여 준다
```

이것들은 echo문에서 사용되였던 단어들과 같다. cat note2가 무엇을 나타내겠는가는 아주 명백하다.

1.10.5 행, 단어, 문자의 계수(wc)

note1파일은 현재 몇개의 행들을 가지고 있는가? wc(word count)지령이 이 질문에 보다 정확한 답을 준다.

```
$ wc note1                wc는 5개 단어들을 보여 준다
       1     5     26 note1
```

이것은 UNIX의 전형적인 동작인데 여기에는 매렬의 의미를 설명해 주는 머리부들이 없다. 이 파일이 1개의 행과 5개의 단어, 26개의 문자들을 가지고 있다는것을 알려면 안내서를 보든지 아니면 이 책을 앞에 놓고 보아야 할것이다. 그러나 wc를 특정한 인수 -1과 함께 사용하여 행들만을 계수할수 있다.

```
$ wc -l note1
      1 note1
```

-l은 물론 wc의 인수이지만 인수가 -로 시작될 때에는 **선택항목**(option)이라고 한다. 많은 지령들과 마찬가지로 wc도 몇개의 선택항목들을 지원한다. 나중에 우리는 파일안의 행들을 계수하는 표준함수를 리용하지 않고 이 지령과 선택항목을 사용하여 어떻게 사용자와 파일들의 수를 계수하는가를 보게 될것이다.

1.10.6 파일속성들의 검사(ls -1)

매 파일은 내용과는 별도로 자기와 협력하는 몇가지 **속성**(attribute 또는 property)들을 가진다. 이 속성들은 파일 그자체에 저장되는것이 아니라 하드디스크의 다른 장소에 저장된다. 파일이름들의 목록을 표시하는것만으로는 때때로 불충분하며 사용자는 이 속성들에 대해 더 알 필요가 있다. -l선택항목을 가진 ls지령이 이 일감을 수행한다.

```
$ ls -1
total 2                  이것을 무시한다
-rw-r-r-  1  romeo  dialout  26 Feb  8:11:17 note1
-rw-r-r-  1  romeo  dialout  27 Feb  8:11:17 note2
```

여기서는 이 파일들의 보다 구체화된 목록을 보여 주고 있다. 한 파일은 26개의 문자를 가지고 있고 다른 파일은 27개의 문자를 가지고 있으며 다같이 romeo가 소유하고 있다(모든 파일들은 그 누구에게인가 소속되여 있다). 파일들은 2월 8일 11시 17분과 11시 18분에 마지막으로 수정되였으며 파일이름들은 마지막렬에 현시되여 있다.

첫렬은 r, w, -문자들의 결합을 사용하는 보다 재미 있는 문자렬을 포함하고 있다.

이 글자들의 결합은 파일을 읽기쓰기할수 있는 사용자들을 결정한다. 소유자는 자기가 소유한 모든 파일들을 읽거나 쓸수 있지만 다른 사람들은 일정한 제한을 받는다. 이와 같은 권한은 특정한 지령(chmod)에 의해 조종되며 제7장에서 이 지령을 사용하게 된다.

이제는 UNIX배경에 대해 어느 정도 알 때가 되였다.

1.11 UNIX의 력사

UNIX가 출현하기전까지 조작체계들은 특정한 기계를 대상으로 하여 설계되였다. 그것들은 **저수준언어**(low-level language: 사람이 해석하기 힘든 코드를 사용한 아쎔블리어와 같은)로 작성되였다. 이 체계들은 속도는 빠르지만 설계가 대상으로 한 그 하드웨어에 국한되여 있었다. 한 체계를 위하여 설계된 프로그람들은 다른 체계에서는 실행되지 않았다. 바로 이것이 AT&T회사의 켄 톰슨과 데니스 리치에가 UNIX체계를 만들어 내던 당시의 콤퓨터산업실태였다.

1969년 AT&T는 련속실행과 원격사용을 내용으로 한 유연조작체계의 개발에 종사하던 MULTICS계획으로부터 자기의 팀을 철수시켰다. 그다음 톰슨과 리치에는 품위 있는 파일체계와 지령해석기(쉘) 그리고 편의프로그람묶음을 가진 자그마한 체계를 설계하고 구축하였다. 하지만 그들이 바란것은 한가지이상의 하드웨어에서 실행되는 일반성 있는 조작체계였다. 1973년에 그들은 리치에가 창안한 **고수준언어**(high-level language)인 C언어로 전반적인 체계를 다시 서술하였다. 이식성은 UNIX의 힘 있는 특징들중의 하나로 되였다.

1.11.1 버클리

AT&T회사는 쏘프트웨어판매에 대한 금지령을 받았으며(그후 이 법은 페지되였다.) 하는수없이 생산물을 대학과 연구소들에 일반값으로 나누어 주고 이에 대한 아무런 보수도 받지 못하였다. 캘리포니아종합대학의 버클리(UCB)에서는 자기 식의 UNIX를 창안하였다. 그들은 이것을 BSD UNIX(Berkeley Software Distribution UNIX)라고 불렀다. 이 판본들은 세계적으로 인기를 끌기 시작하였으며 특히 대학과 공학계에서 더욱 통속화되였다. 그후 UCB는 UNIX에 대한 개발을 일체 단념하였다.

버클리는 AT&T뒤에 생겼던 공백을 메꾸어 주었으며 그후 자기들대로 모든 조작체계를 다시 작성하기로 결정하였다. 그들은 UNIX체계의 표준편집기(vi)를 창안하였으며 통속적인 쉘(C쉘)을 창안하였다. 버클리는 또한 보다 더 좋은 파일체계와 더 만능인 우편특징 그리고 보다 더 좋은 파일련결방법(기호련결)도 창안하였다. 그후 그들은 망작업규약쏘프트웨어(TCP/IP)를 제공함으로써 인터네트를 가능하게 해주었다. AT&T와 같이 그들도 이것을 많은 회사들에 무상으로 제공해 주었다.

1.11.2 기타

싼은 BSD체계를 자기의 UNIX상표(SunOS로 되였다.)를 발전시키기 위한 발판으로서 사용하였다. 오늘날 그들의 UNIX판본은 Solaris로 알려 져 있다. 다른 회사들도 자기의 고유한 상표를 가지고 있다. IBM은 AIX를, HP는 HP-UX를 그리고 DEC는 Digital UNIX를 만들어 냈다. 표 1-3에 UNIX의 유명한 변종들을 보여 준다.

표 1-3. UNIX의 변종들

제 품	개발회사	비 고
Xenix	Microsoft Corporation	현재 중지되였다
BSDi, BSD/OS, Free BSD	Berkeley Software Design	BSD UNIX를 만든 일부 사람들에 의하여 시작된 상업적시도
Sun OS	Sun Microsystems	지금은 독자적으로 존재하지 않으며 Solaris와 합쳐 졌다
Solaris	Sun Microsystems	PC에서 실행되는 판본을 가지고 있다
HP-UX	Hewlett-Packard	
AIX	IBM	
Ultrix	DEC	현재 중지되였으며 Digital UNIX로 교체되였다
Digital UNIX	DEC	
IRIX	Silicon Graphics	
SCO Open Server	Santa Cruz Operation	현재 중지되였으며 SCO UnixWare로 교체되였다
SCO UnixWare	Santa Cruz Operation	PC상에서 실행한다(Open Server와 같이)

매 제작자들이 UNIX를 수정 및 확장하여 자기들의 판본을 만들어 냄으로써 본래의 UNIX는 독자적인 생산물로서의 본성을 잃어 버리게 되였다. BSD제품들은 System V제품들과 많이 달라 졌으며 끊임없이 갱신되였다. UNIX로 불리우는데 충분한 생산물로서 표준들이 개발되고 있을 때 AT&T는 BSD제품을 기본적으로 개정할것을 결정하였으며 종당에는 자기들의 System V 3.2와 BSD, Sun OS, XENIX변종들을 마지막판본인 Sytem V Release 4(SVR4)에 통합하였다.

1.11.3 인터네트

SVR4가 출현하기전에도 미국방성의 한 팀인 DARPA는 개발회사들을 채용하여 콤퓨터기술을 리용한 믿음직한 통신체계를 개발하였다. 빈톤 커프(Vinton Cerf)와 로버트 칸(Robert Kahn)에 의하여 DARPA의 ARPANET망은 **파케트절환**(packet-switching)기술로 작업할수 있도록 꾸려 졌다. 이 씨나리오에서 자료는 파케트들로 쪼개지며 이 파케트들은 서로 다른 통로들을 택할수 있고 정확한 순서로 재조립된다. 이것은 통신에서 인터네트가 사용하는 규약들의 묶음인 **TCP/IP**를 탄생시켰다.

DARPA는 UCB에게 BSD UNIX에서 TCP/IP를 실현할것을 의뢰하였다. ARPANET는 1983년에 TCP/IP로 전환되였다. 같은 해에 버클리는 TCP/IP를 구축한 첫 UNIX판본을 발표하였다. 콤퓨터과학연구계는 모두 UNIX를 사용할수 있게 되였으며 망은 마치 번개불처럼 삽시에 퍼져 나갔다. TCP/IP를 UNIX에 통합시킨것과 그것을 개발기초로 사용한것은 **인터네트**(와 UNIX)의 급속한 장성의 주요요인들로 되였다.

1.11.4 Windows의 위협

그동안 마이크로쏘프트는 Windows로 비대해 졌다. 도형사용자대면부(GUI:graphical user interface)는 일감을 실행하는데서 리해하기 힘들고 복잡한 지령선택항목들대신에 마우스를 사용하였다. 이 선택항목들은 내리펼침차림표의 칸들과 단일선택단추(radio button)들로부터 선택될수 있으며 이것은 기초적인 조작체계기능을 보다 쉽게 조종할수 있게 해준다. Windows는 처음에 탁상용콤퓨터시장을 휩쓸었으며(Windows 3.1/95/98로) 그다음에는 UNIX가 오랜 기간 지배해 온 봉사기시장에 치명적인 타격을 주었다(Windows NT로).

UNIX가 자기의 생존을 위해 Windows형의 대면부를 절실히 요구하고 있을 때 매써츄쎄쯔기술연구소(MIT)는 UNIX에 있어서 첫 창문체계로 되는 X Window를 도입하였다. X Window는 Microsoft Windows의 중요특징들중 많은것들을 가지고 있으며 다른것들도 많이 가지고 있다. UNIX의 모든 변종들은 현재 X외에 파일 및 등록부들을 조종할뿐아니라 체계구성파일들을 갱신하는 수많은 다른 도구들도 가지고 있다.

결국 UNIX의 힘은 모든 지령들과 그것들의 여러가지 선택항목들로부터 산출된다. 그 어떤 도형대면부도 정교한 정합구조를 리용하여 모든 종류의 속성들을 다 가지고 있는 파일들을 찾아 내는 find지령을 전혀 대신하지 못한다.

1.11.5 UNIX표준들과 POSIX

지난 몇해사이에 몇가지 중요한 발전들이 이룩되였다. 1992년에 AT&T의 UNIX기업은 Netware라고 부르는 망작업쏘프트웨어를 만들어 낸 노벨(Novell)에 팔리웠다. 그후 노벨은 X/OPEN이라고 부르는 표준단체에 UNIX상표를 넘겨 주었다. X/OPEN은 그후 The Open Group에 병합되였으며 이 The Open Group이 현재 UNIX표준을 소유하고 있다. 1997년에 The Open Group은 **단일UNIX명세**(Single UNIX Specification)를 출판하였으며 그 다음해에는 **UNIX98**을 출판하였다.

지금은 하나의 표준조작대면부묶음이 UNIX의 개발을 안내하고 있다. 이 묶음을 **이식가능조작체계대면부**(POSIX: Portable Operating System Interface)라고 부르고 있으며 이것은 UNIX에 기초하고 있다. 사람들은 코드를 전반적으로 재작성하지 않고 서로 다른 UNIX체계들을 지나 자유롭게 이동할수 있는 응용프로그람들을 개발할것을 바라고 있었다. POSIX가 일반적인 조작체계표준을 설정한다고 하여도 어쨌든 UNIX에 기초하고 있다.

POSIX대면부들은 IEEE의 요구에 따라 개발되였으며 한개 표준묶음을 구성하고 있다. POSIX.1은 C응용프로그람대면부를 제공한다. POSIX.2는 쉘(이 책에서 언급된)과 편의프로그람들을 취급한다. 현재는 두가지가 다 단일UNIX명세에 포함되여 있다. 대다수 UNIX판매자들은 현재 The Open Group과 적극적으로 협력하고 있으며 UNIX표준에 기초하여 생산물을 만들어 내고 있다. UNIX는 열린 체계이다.

1.12 Linux와 GNU

UNIX가 최종적으로 실용화되였음에도 불구하고 리챠드 스톨맨과 리누스 토발즈는 다른 구상을 하였다. 토발즈는 콤퓨터세계를 폭풍처럼 휩쓴 자유로운 UNIX 즉 Linux의 아버지이다. 스톨맨은 자유쏘프트웨어재단(이전에는 GNU라고 하였으며 GNU는 영어로 《GNU's Not Unix(GNU는 유닉스가 아니다.)》를 나타내는 재귀적인 략어이다.)를 운영하고 있다. 많은 Linux의 중요도구들이 GNU에 의해 작성되고 무상공급되였다.

Linux는 개발자들과 판매자들이 원천코드를 공개하도록 한 GNU일반공증허가에 따라 배포되였다. Linux는 특히 망작업과 인터네트기능들에서 힘 있으며 인터네트봉사기나 구역인터네트설정에서 비용이 매우 효과적으로 들도록 해주고 있다. 오늘 Linux에 대한 개발은 자유쏘프트웨어재단(Free Software Foundation)의 지령에 따라 세계 여러 곳에서 수행되고 있다.

가장 인기 있는 Linux변종들은 Red Hat, SuSE, Caldera이다. 여러장의 CD-ROM에 적재한 이러한 변종들에는 C 및 C++콤파일러들로부터 Java에 이르기까지 그리고 perl, phthon, tcl과 같은 해석기들, Netscape와 같은 열람프로그람들, 인터네트봉사기들과 다매체쏘프트웨어와 같은 수많은 쏘프트웨어가 포함되여 있다. 이것은 다해서 50딸라밖에 안되며 대부분은 인터네트로부터 무상으로 내리적재(download)된다. 주요콤퓨터판매자들(마이크로쏘프트를 제외한)모두가 Linux를 지원할것을 약속하였으며 대부분이 자기의 쏘프트웨어를 이 가동환경에 이식하였다. 이 책에서는 Linux에 대해서도 론의하였다.

1.13 UNIX의 내부

얼마 안되는 본질적으로 단순하면서도 추상적인 개념들이 UNIX체계전반을 지원하고 있다. 톰슨과 리치에[*1]가 말한바와 같이 UNIX의 성공은 그것이 《새로운 발명품이라기보다는 오히려 풍부한 착상들에서 심중히 골라 충분히 채취한것이라는데 있으며 특히는 그것들이 작으면서도 강력한 조작체계를 실현할수 있는 열쇠로 된다는데》 있다. UNIX는 더이상 작은 체계가 아니며 확고하게 강력한 체계이다. 이제는 그 기초에 대해 보기로 하자.

1.13.1 핵심부와 쉘

이 풍부한 착상들가운데서 가장 중요한것은 2개의 매개물들인 **핵심부**(kernel)와 **쉘**(shell)사이의 작업분할이다. 쉘은 사용자와 대화하며 핵심부는 기계의 하드웨어와 대화한다. 이 둘사이의 관계에 대해서는 1.1에서 폭 넓게 설명하였으며(이름은 주지 않고) 또한 그림 1-1에서 그 관계에 대해 보여 주고 있다.

[*1] Ritchie, D.H.,and K. Thompson. 《the Unix Time-Sharing System.》 The Bell System Technical Journal Vol. 57 No.6,July-August, 1978.

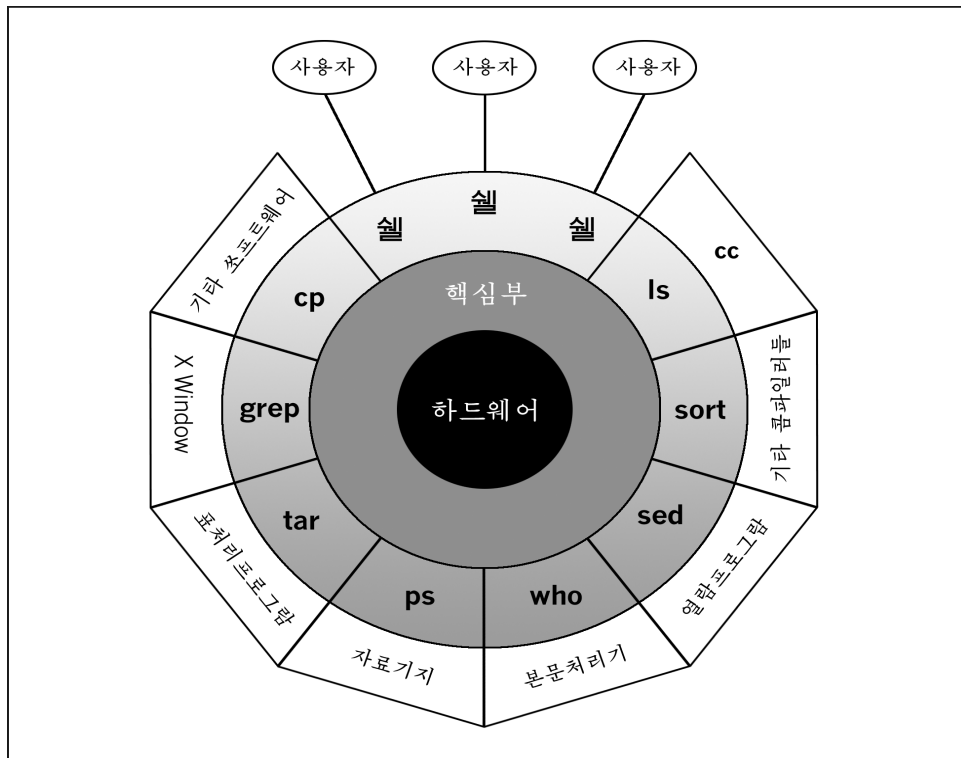



그림 1-1. 핵심부-쉘-사용자사이의 관계

핵심부는 조작체계의 중심이며 거의나 C로 서술된 프로그람들의 집합으로서 하드웨어와 직접 통신한다. 이것은 체계가 기동할 때 기억장치에 넣어 지는 UNIX체계의 한 부분으로서 체계자원을 관리하고 사용자들과 프로쎄스(process)들사이에 시간을 할당해 주며 프로쎄스의 선후차를 결정해 주고 사용자가 귀찮아 하는 기타 다른 과제들도 맡아 수행한다. 다른 프로그람(응용프로그람)들도 **체계호출**(system call)이라고 부르는 어떤 함수묶음을 통하여 핵심부의 봉사에 접근한다. 핵심부는 사실상 콤퓨터자원에로 가는 프로그람의 관문(gateway) 즉 조작체계라고 할수 있다.

콤퓨터에는 본래 지령들을 번역하여 작용시키는 능력이 없다. 이를 위해서는 해석기가 있어야 하며 UNIX체계안에서 이 일감은 조작체계의 바깥부분인 쉘이 조종한다. 쉘은 사용자로부터 지령을 접수하여 특수한 문자들을 해독하며 이 문자들을 찾아서 간략된 지령행으로 재구성한다. 쉘은 최종적으로 핵심부와 통신하여 지령이 실행되였는가를 알아 본다. 실제적으로 이것은 사용자와 핵심부사이의 대면부이며 사용자를 핵심부기능들에 대한 지식으로부터 격리시킨다.

주해

핵심부는 /stand/unix, /unix, /kernel/genunix(Solaris)파일들에 의해 표현된다. 쉘은 /bin/sh(Bourne쉘), /bin/csh(C쉘), /bin/ksh(Korn쉘)에 의하여 표현된다. 기동시에 체계초기적재프로그람은 핵심부를 기억장치에 적재한다. 사용자가 가입할 때에는 이 쉘들중 하나가 실행되여 봉사해 준다. 실행되고 있는 쉘을 알기 위해서는 echo $SHELL지령을 사용한다.

Linux

핵심부는 /boot/vmlinuz파일에 의해 표현되며 쉘은 /bin/bash(bash-표준Linux쉘)에 의하여 표현된다.

1.13.2 다중사용자체계

한개 체계에서 여러 사용자들이 작업한다는 개념은 종종 Windows사용자들을 당황케 한다. Windows는 본질에 있어서 기억기와 CPU, 하드디스크가 다같이 한명의 사용자에게 복무하는 단일사용자체계(single-user system)이다. UNIX에서는 실질적으로 그 자원들이 모든 사용자들사이에서 공유된다. 즉 UNIX는 **다중사용자체계**(multiuser system)이다. 다중사용자기술은 모두를 상대로 하는 거대한 사회자라는데 있다.

기만적인 효과를 얻기 위해 콤퓨터는 단위시간을 여러개의 토막으로 쪼개고 매 사용자에게 한토막씩 할당한다. 따라서 기계는 임의의 시각에 어느 한 사용자의 일감을 수행하고 있을것이다. 할당된 시간이 지나는 순간 이전에 수행되던 일감은 정지되고 다음사용자의 일감이 시작되게 된다. 이 과정은 시계가 한 바퀴를 완전히 돌아 올 때까지 계속되며 그다음에는 처음사용자의 일감이 다시 시작되게 된다. 이와 같은것을 핵심부는 1초동안에도 여러번 수행한다. 이 모든것은 프로쎄스관리체계에 의해 조종되는데 이 체계에 대해서는 제10장에서 설명하였다.

1.13.3 다중과제처리체계

UNIX에서는 또한 한명의 사용자가 여러개의 과제들을 동시에 실행할수 있다. 한명의 사용자가 한 파일을 편집하면서 인쇄기에서 또 다른 파일을 인쇄하며 친구에게 전자우편을 보내고 WWW를 열람할수(어느 응용프로그람도 완료하지 않고) 있다. 핵심부는 사용자의 다중요구를 조종할수 있게 설계되여 있다. 이와 같은 **다중과제처리**(multitasking)환경에서는 오직 하나의 일감만이 전경에서 실행되며 나머지일감들은 배경에서 실행된다. 사용자는 전경과 배경사이에 일감들을 절환할수 있으며 그것들을 중지 또는 완료할수 있다. 다중과제처리역시 프로쎄스관리체계의 중요한 구성요소로서 이에 대해서는 제10장에서 론의하게 될것이다.

주해

Windows 역시 사용자가 여러 창문들에서 동시에 작업할수 있는 다중과제처리체계이다. 언제나 제목띠가 강조되여 있는 한개 창문만이 전경에서 작업한다.

1.13.4 한가지를 수행한다

설계자들은 결코 지나치게 많은 기능들을 몇개의 도구들에 묶어 놓으려고 시도하지 않았다. 그대신 《한가지를 수행한다(do one thing well)》는 방식은 매개가 단일일감들을 수행하는 수백개의 지령들을 개발하도록 해준다. 이전에 설명한바와 같이 모든 UNIX도구들(몇개를 제외하고)은 혼자서는 힘이 없다. 하지만 **프로쎄스간통신**(interprocess communication)기능을 사용하여 쉘은 한 지령이 다음지령에, 또 그다음지령에 자료를 전달하도록 배렬할수 있다.

여러개의 도구들을 호상 련결시켜 그것들의 결합적사용을 얻을수 있다. 때문에 모든 문제들을 혼자서 풀려고 시도하는 지령보다 특정한 하나의 기능을 조종하는 지령을 가지는것이 더 낫다. UNIX는 이러한 개념으로부터 출발하였지만 그후 체계에 도구들이 첨부되면서 이에 대해 왕왕 잊어 버리게 되였다. UNIX의 장치독립기능들에 대해서는 제8장에서 취급된다.

1.13.5 광범히 리용할수 있는 파일

카르 크리스챤[*2]은 UNIX체계에서 2개의 강력한 개념들을 발견하였다. 즉 《파일은 장소를 가지며 프로쎄스는 생명(살아 움직이는)을 가진다.》 첫번째것은 파일들은 공간에 배치되며 그 공간은 예정된 장소에

[*2] Kaare Christian. The UNIX Operating System. New York : John Wiley, 1988.

그것들이 쉽게 놓일수 있게 하여 준다는것을 의미한다. 더우기 파일체계안에서 정해 진 장소에 배치될수 있으며 한 장소에서 다른 장소에로 옮겨 질수 있다. 이와 같이 현실적으로 살아 움직이는 모형에 의하여 UNIX파일체계는 광범히 리용되게 되였다. 프로쎄스의 역할에 대해서는 1.13.2와 1.13.3에서 설명하였다.

UNIX는 사실 사용자가 사용하고 있는 파일의 형식에 대해서는 알려고 하지 않는다. UNIX는 등록부와 장치들도 파일체계의 성원들로 간주하고 있다. UNIX에 있어서 파일은 바이트들의 배렬이며 실질적으로 모든것 즉 본문, 목적코드, 등록부구조를 포함할수 있다. 지배적인 파일형태는 본문이며 체계의 동작은 주로 본문파일들에 의해 조종된다. UNIX는 편집기를 사용하지 않고 이 파일들을 편집할수 있는 본문조작도구들의 거대한 묶음을 제공해 주고 있다. 파일체계와 파일속성들에 대해서는 제6장과 제7장에서 론의하게 된다.

1.13.6 패턴정합

UNIX는 사용자의 타자부담을 줄이는데 도움이 되는 패턴정합기능을 제공한다. *와 같은 문자들이 문자렬에 첨부되여(chap*와 같이) 여러개의 파일이름들을 정합하는데 사용될수 있다. 파일이름들을 심중히 선택한다면 사용자는 간단한 표현을 써서 파일이름전체에 접근할수 있다. 파일들과는 별도로 특수한 지령묶음이 *와 같은 문자들에 의하여 일반화된 패턴(**정규식**이라고 부른다.)을 사용한다. 이 책에서는 정규식들의 중요성에 대해 강조하면서 그것들을 리용하여 사용자가 복잡한 패턴정합과제들을 어떻게 수행하는가에 대해 보여 준다. 이 문자들로 파일이름들을 정합하는데 대해서는 제8장에서 취급하고 있다. 제15장은 정규식들에 대해 폭 넓게 담고 있다.

1.13.7 프로그람작성기능

UNIX쉘은 프로그람작성언어이기도 하다. 즉 UNIX쉘은 림시적인 말단사용자가 아니라 프로그람작성자를 위하여 설계되였다. UNIX쉘은 조종구조들과 순환들, 변수들과 같은 필요한 모든 구성요소들을 가지고 있는것으로 하여 위력한 프로그람작성언어로 작성되였다. 이 기능들은 UNIX지령들까지도 자기 문법에 포함시킨 프로그람들인 **쉘스크립트**(shell script)들을 설계하는데 리용된다. 쉘스크립트들은 본문조작과제에 아주 유용하다.

이 쉘스크립트들을 사용하여 많은 체계기능들을 조종하고 자동화할수 있다. 만일 체계관리를 직업으로 하고 싶다면 쉘의 프로그람작성기능들에 대해 잘 알아야 한다. 유능한 UNIX프로그람작성자들은 본문조작문제에 관하여 그 어떤 다른 언어(perl은 제외)에 의거하려 하지 않는다. 쉘프로그람작성에 대해서는 제18장과 제19장에서 취급하고 있다.

1.13.8 이식성과 체계호출

UNIX는 C로 서술된다. 특정한 기능을 조종하는 수천개의 지령들이 있지만 이것들은 모두 **체계호출**(system call)이라고 부르는 몇개의 함수들을 사용한다. 이 호출들은 핵심부안에 구축되여 있으며 모든 서고함수들과 편의프로그람들이 그것들을 사용하여 서술된다. 모든 UNIX변종들은 한가지 공통점을 가지고 있는데 그것은 그들모두가 꼭 같은 체계호출을 사용한다는것이다. 만일 서로 다른 체계호출을 사용한 조작체계가 있다면 그것은 UNIX가 아니다.

체계호출을 통한 대화는 체계와의 효과적인 통신수단을 표현한다. 프로그람작성자는 쓰기조작을 수행하는 프로그람내부에 들어 가지 않고 write()체계호출을 사용하여 파일쓰기를 진행한다. 같은 체계호출은 자기의 도구를 개발하려는 모든 C프로그람작성자들에게 유용하다. 이것은 또한 임의의 UNIX체계에서 개발된 쏘프트웨어는 다른 UNIX기계에 쉽게 이식될수 있다는것을 의미한다. 프로그람재작성에 필요한것은 체계호출 그자체가 아니라 체계호출의 수행부분이다.

1.13.9 문서

UNIX문서(documentation)는 지난 시기와는 달리 더이상 다루기 어려운 부분이 아니다. 이따금 순탄치 않는 때가 더러 있기는 하지만 대부분 쉽게 다룰수 있다. 기본적인 직결도움말기능은 man지령인데 거기에는 지령들과 그 구성파일에 대한 가장 중요한 참고내용들이 들어 있다. 별도로 문서를 가지고 있지 않는 UNIX란 없다. UNIX문서와 도움말기능에 대해서는 제2장에서 론의한다.

이와 같이 거대한 본문자료기지가 있고 인터네트를 통한 양성과정과 연구회, 토론회들을 진행할수 있은것으로 하여 UNIX는 불과 5년전에 비해 오늘날 더 편리한것으로 되고 있다.

기계와의 관계를 보다 편리하게 구성하는데 성공하게 되자 톰슨과 리치에는 이와 같은 성공으로부터 남이 아니라 자기자신들의 사용을 위한 설계를 하였다. 그들이 이렇게 한것은 UNIX가 초기부터 상업적 생산물로 개발되지 않았고 미리 정의된 대상을 가지지 않았기때문이였다.

요 약

조작체계는 기계의 하드웨어에 대한 접근을 요구하는 응용프로그람들과 대화한다. 조작체계는 또한 지령언어해석기를 사용하여 지령들을 보내오는 사용자와도 대화한다. 조작체계는 프로그람실행과 기억기할당, 프로그람오유들을 조종한다.

UNIX는 여러 사용자들에 의해 동시에 사용될수도 있으며 단독적인 워크스테이션에서 사용될수도 있다. 사용자는 체계관리자에게서 배당 받은 사용자이름과 통과암호를 입력하여 UNIX체계에 들어 간다. 보안을 위하여 통과암호는 화면에 현시되지 않는다. 사용자는 [Ctrl-d]를 누르거나 logout지령을 사용하는 방법으로, 또는 exit를 입력하여 체계로부터 탈퇴할수 있다.

UNIX는 $나 %와 같은 몇개의 프롬프트들을 특징 지어 준다. 사용자는 이 프롬프트에 적당한 지령을 입력할수 있다. UNIX지령들은 소문자로 되여 있으며 대문자파일이름과 소문자파일이름을 별개의 파일로 따로따로 취급한다.

사용자는 passwd지령으로 자기의 통과암호를 변경시킬수 있다. who를 리용하여 사용자들의 목록을 현시할수 있으며 tty로는 사용자의 말단이름을 알수 있다.

SHELL과 TERM은 체계안에서 2개의 중요한 변수들이며 $접두사를 가진 변수이름으로 echo를 사용하여(실례로 echo $SHELL) 그 변수들의 값을 얻을수 있다. set는 모든 체계변수들의 목록을 보여 준다.

오유가 만들어 졌을 때 사용할수 있는 몇개의 건조작들이 있다. 이에 대해서는 표 1-2에 보여 준다.

많은 지령들은 인수라고 부르는 추가적인 단어들과 함께 사용할수 있다. 기호 -로 시작하는 특수한 인수들을 선택항목이라고 한다.

mkdir를 리용하여 등록부를 만들수 있고 cd로 그것을 변경시킬수 있으며 pwd로는 현재등록부를 검사해 볼수 있다.

ls는 어떤 등록부의 파일목록을 현시하며 -1선택항목과 함께 사용될 때에는 파일들의 속성을 보여준다. echo지령이 >기호와 함께 사용될 때에는 파일안에 본문을 배치한다. cat는 파일의 내용을 현시하며 wc는 행과 단어 및 문자들의 수를 계수한다.

UNIX는 이식가능한 조작체계를 만들 목적으로 AT&T벨연구소들에서 켄 톰슨과 데니스 리치에에 의하여 개발되였다. UNIX는 C로 서술되였으며 현재 넓은 범위의 기계들에서 실행되고 있다.

또한 개발작업의 많은 부분들이 캘리포니아종합대학의 버클리에서 진행되였으며 여기서 BSD UNIX를 만들었다. 버클리는 C쉘, vi편집기 그리고 망작업편의프로그람의 TCP/IP묶음을 담당하고 있다.

AT&T는 자기의 판본을 일부 다른 판본들과 통합하여 SVR4(System V Release 4)를 발표하였다. 벨연구소들과 버클리는 그후 UNIX에 대한 작업을 중지하였으며 현재 UNIX는 The Open Group의 상표로 되였다.

역시 대중화되여 가고 있는 Linux는 자유쏘프트웨어를 작성, 배포하는 GNU로부터 기증되는 자유로운 UNIX이다. Linux는 인터네트와 관련한 많은 기능들을 가지고 있다.

작업분담은 하드웨어를 직접 다루는 핵심부와 사용자와의 대화를 수행하는 쉘에 의해 분배된다. 쉘은 입력된 지령을 처리하고 특수문자들을 주사하여 핵심부가 리해할수 있는 형식으로 그것을 재구성한다. 핵심부는 기억기안에 상주하며 unix나 vmlinuz파일에 의하여 표현된다.

여러 사용자들이 이 체계를 함께 사용할수 있으며 한명의 사용자가 여러개의 일감들을 동시에 실행할수도 있다.

매 UNIX지령은 하나의 단일한 일감을 수행한다. UNIX는 사용자가 이 기초적인 구성블로크들을 련결하여 복잡한 지령루틴들을 구성하게 한다.

UNIX파일들은 개념적으로 단순하다. UNIX는 등록부와 장치들을 파일로 간주한다. 본문파일들은 체계동작을 조종하고 UNIX는 편집기를 사용하지 않고 그것들을 변경시킬수 있는 넓은 범위의 도구들을 가지고 있다.

UNIX는 패턴정합을 위한 폭 넓은 기능들을 특징 지어 주며 체계호출서고와 도형사용자대면부를 가지고 있다. UNIX문서는 폭이 매우 넓다.

시험문제

1. 조작체계와 대화하는 두 매개물은 무엇인가?
2. 매 문자들은 그와 관련된 수를 가진다. 이것을 무엇이라고 부르는가?
3. 다른 건들과 항상 함께 사용되는것은 어떤 건들인가?
4. mailserver login과 같은 프롬프트를 보게 되는 경우 mailserver가 무엇을 표시하고 있다고 생각하는가?
5. 통과암호를 사용하지 않고 체계에 가입할수 있는가?
6. 만일 체계가 Login incorrect라고 표시하면 이것은 사용자의 가입이름이 틀렸다는 뜻인가?
7. 어느 건으로 지령을 중단시키겠는가?
8. 사용자가 체계에 가입할 때 사용했던 사용자이름을 잊어 버렸다. 이것을 어떻게 찾아 낼수 있는가?
9. 어느 지령으로 파일들의 구체적인 목록을 볼수 있는가?
10. cat지령은 어디에 사용되는가?
11. UNIX조작체계의 기본설계자는 누구인가?
12. AT&T는 왜 UNIX를 무료로 배포하였는가?
13. BSD UNIX는 어디서 시작되였는가?
14. 싼의 UNIX판본은 무엇이라고 알려 져 있는가?
15. 어느 UNIX변종들이 PC에서 실행되는가?
16. 오늘날 UNIX상표를 소유한것은 누구인가?
17. Linux뒤에 있는 두명의 학자들은 누구인가?
18. GNU일반공중허가와 관련한 특이한 특징은 무엇인가?
19. 체계가 시동되자마자 조작체계의 어느 부분이 기억기에 적재되는가?

20. 왜 UNIX가 다른 조작체계보다 더 이식가능하다고 하는가?
21. UNIX를 두개의 주요파로 나눌수 있는가? 싼회사의 UNIX는 어느 파에 속하는가?
22. System V에서와 Linux에서 핵심의 이름은 각각 무엇인가?
23. 다중과제란 무엇을 의미하는가?
24. 왜 UNIX도구들은 복잡한 일감보다 단순한 일감들을 실행하게 되는가?
25. UNIX의 창문화체계는 무엇이라고 알려 져 있는가?
26. UNIX체계들에서 유용한 해석언어 몇개를 말하시오.
27. 3개의 유명한 Linux변종들을 지정하시오.

련습문제

1. 자기 친구의 이름을 통과암호로 사용할수 있는가? 만일 할수 있다면 그 이름을 사용하겠는가?
2. 체계에서 탈퇴하기 위하여 차례로 시도해 보아야 하는 3개의 지령은 어느것인가? 그중 언제나 동작하는 지령은 어느것인가?
3. 2개의 지령 즉 finger와 users를 따로따로 입력하시오. 그것들의 출력에서 어떤 차이점을 보게 되는가?
4. 만일 [Backspace]건이 없거나 이 건은 있지만 종전의 본문을 지우지 못할 때 어느 건을 사용하겠는가?
5. [Enter]건이 고장난 경우 어떻게 지령을 계속 입력하겠는가?
6. 화면출력이 흐르는것을 어떻게 정지시키는가? 계속하자면 어떻게 해야 하는가?
7. 다음의 2개 지령들을 시험해 보시오. 무슨 일이 일어 났는지를 어떻게 알수 있으며 어떻게 결속하겠는가?

   ```
   echo > README[Enter]
   echo > readme[Enter]
   ```

   TERM과 SHELL은 무엇인가? 이것들의 값을 어떻게 얻는가?
8. tty지령과 TERM은 어떻게 다른가?
9. 사용자가 지령을 틀리게 입력하였으며 프롬프트가 돌아 오지 않았다. 원인은 무엇이며 정상으로 재생시키기 위해 어떤 건조작을 시도해 보겠는가?
10. 만일 사용자의 말단이 낯선 출력을 만들어 내게 되면 최종적으로 어느 지령을 시험해 볼수 있는가? 만일 [Enter]건도 동작하지 않는 경우 그 지령을 어떻게 실행시키겠는가?
11. 일감이 1시간동안 실행되고 있는데 이 일감은 중요하기때문에 중단시키지 말아야 한다. 이때 이 일감을 정지시키고 다른 일감을 실행한 다음 다시 첫번째 일감을 계속할수 있는가?
12. echo > foo.bar.gz로 이 파일을 만들어 보시오. 그렇게 할수 있는가?
13. 2개의 파일이름을 인수로 하여 wc지령을 사용하시오. 어떤 차이점들이 있는가?
14. 등록부를 만들고 거기로 변경시키시오. 그다음 새 등록부에 또 다른 등록부를 만들고 또 그 등록부에로 변경시키시오. 이제는 그 어떤 인수도 없이 cd를 실행시키고 뒤따라 pwd를 실행시키시오. 어떻게 결속하겠는가?
15. 쉘을 왜 지령해석기라고 부르는가?
16. 등록부들과 장치들, 말단들, 인쇄기들에 일반적인것이 한가지 있는데 그것은 무엇인가?
17. UNIX체계가 다른 기계에로 옮겨 질 때 변경되여야 하는것이 무엇인가?

제 2 장. UNIX지령에 대한 리해

UNIX지령계는 매우 방대하며 20년이상 구성되여 왔다. 체계에는 수백개의 지령들(때로는 수천개)이 있다. 대다수의 지령들은 단순일감을 수행하며 많은것들이 간결한 문법을 사용하고 짧은 통보문을 처리한다. 만일 지령을 틀리게 입력하면 체계는 오유통보문을 간단히 보여 주든가 전혀 보여 주지 않을수도 있다. Windows프로그람들과는 달리 UNIX지령들은 대화식이 아니므로 일부 지령들은 처음에 사용하기 어렵다.

UNIX설계자들은 사용자가 이 지령들중 대부분의 문법과 선택항목들을 기억하지 못할것이라는데 대하여 인정하였다. 때문에 그들은 몇개의 대규모적인 문서를 제공하였다. 이 장에서는 일반화된 UNIX지령문법을 면밀하게 검토하고 그 구성요소들인 선택항목들과 인수들, 기타 파라메터들의 의미에 대해 리해하게 된다. 또한 체계에서 리용할수 있는 직결도움말기능들을 사용하는 방법에 대해서도 배우게 된다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 대소문자를 구별하며 확장자가 없는 지령의 일반적특징들에 대해 인식한다(2.1).
- PATH변수가 지령들의 위치를 어떻게 알아 내는가에 대하여 배운다(2.2).
- 내부지령과 외부지령의 차이점을 파악한다(2.3).
- 지령을 인수와 선택항목들로 가르고 있을수 있는 가능한 변화들에 대하여 알아 본다(2.4).
- 지령사용법의 유연성에 대해 배운다(2.5).
- man지령을 사용하여 UNIX문서를 열람한다(2.6).
- 문서의 구조 특히 문법을 설명하는 방법에 대하여 배운다(2.7).
- info지령을 사용하여 Texinfo문서를 본다(2.8).
- apropos와 whatis지령들을 사용하여 일감을 수행할 지령을 식별한다(2.9).

2.1 지령의 일반적특징

UNIX지령은 일반적으로 영어문자들을 사용한 한개의 단어로 구성된다. 설계자들은 UNIX를 순수 자기들이 사용하기 위해 만들었으므로 최소의 건반조작으로 최대의 작업량을 수행할수 있도록 하려고 시도하였다. 그렇기때문에 원래의 UNIX지령들은 대부분이 4문자미만이다. 우리는 이미 ls, cat, who 등 일부 지령들을 사용하였다. 최근의 일부 지령들은 긴 단어들로서 가끔 수자와 밑선기호를 포함하고 있다.

때때로 지령이름을 통해 그 지령의 의미를 짐작할수 있으나 일반적으로는 그렇지 못하다. 실례로 우리는 직관적으로 ls가 목록(list)을 나타내고 있으며 cp는 복사(copy)를 의미한다고 생각할수 있다. 하지만 grep가 패턴(pattern)을 탐색하며 sed가 파일에서 치환(substitution)을 수행한다고 짐작할수 있는가?

본질에 있어서 지령들은 프로그람(주로 C로 서술된)들을 표현하는 파일들이다(파일들에 대해서는 제6장에서 언급된다). 이 파일들은 대체로 등록부라고 부르는 일정한 장소에 저장된다. 실례로 ls지령은 등록부 /bin에 위치하고 있는 어떤 파일이 표현하는 프로그람이다.

UNIX지령파일은 .exe 또는 .com과 같은 특수한 확장자를 가질 필요가 없다. 하지만 원한다면 그것을 제공할수 있다. 길이에서는 거의나 제한이 없으며 지령이름은 255문자까지 쓸수 있다. 이것이 파일체계에서

임의의 파일에 설정되는 한계점으로 된다. 그러나 보다 낡은 UNIX체계들에서는 14문자로 제한되여 있었다.

앞에서는 UNIX가 대소문자를 구별한다는데 대해서도 설명하였다. 지령 ls는 LS와 같지 않다. 만일 ls대신에 LS를 입력하게 되면 체계는 다음과 같이 대답한다.

```
$ LS
sh: LS: not found
```

UNIX체계에는 명백히 LS라는 이름을 가진 지령이 없다. 가령 사용자에게 모든 지령에 대문자를 사용하는 버릇이 붙어 있다면 이와 같은 버릇은 떼버려야 한다. 가장 좋은 방법은 건반에 있는 [Caps Lock]건이 대문자를 발생시키지 않게 설정되여 있는가에 대해 확인해 보는것이다.

UNIX는 LS지령을 제공해 주지 않지만 LS라는 이름을 가진 지령을 만드는데는 방해로 되지 않는다. 체계안에서는 모든 지령들이 파일로서 표현되므로 사용자는 파일 LS를 만들어 파일체계안의 적당한 위치에 배치해야 한다.

2.2 지령들의 위치알아내기(PATH)

지령이 적당한가, 적당치 못한가 하는것을 UNIX는 어떻게 알아 내는가? 지령을 입력하면 체계는 지정된 등록부들에서 그 파일을 탐색한다. 이 등록부들가운데서 해당 파일을 찾게 되면 그것을 실행하지만 만일 찾지 못하게 되면 앞에서 보여 준 실례에서와 같은 통보문을 즉시 내보낸다.

UNIX의 기능은 몇개의 변수들에 의하여 조종된다. UNIX는 자기의 변수들중 한개로부터 탐색할 등록부들의 목록을 얻게 되는데 이 변수를 PATH라고 부른다. 이 변수의 값을 평가하면 두점으로 구분된 등록부목록이 현시된다.

```
$ echo $PATH
/bin:/usr/bin:/usr/x116/bin:/oracle/bin:.              C쉘에서는 출력이 약간 다르다
```

Windows사용자들도 탐색경로를 지정하기 위해 AUTOEXEC.BAT에서 이 이름으로 된 변수를 사용하며 다만 구분문자가 다르다(UNIX에서는 :이며 Windows에서는 ;이다). 이 목록에는 5개의 등록부가 있는데 사용자가 지령을 입력하면 체계는 지정된 순서대로 이 목록을 탐색하여 파일의 위치를 알아내고 그것을 실행시킨다. 이것은 처음에는 /bin을, 그다음에는 /usr/bin을, 또 그다음에는 다음등록부들을 계속 탐색하여 나간다는것을 의미한다. 여기서 마지막의 점(.)은 당분간 무시하기로 하자.

그런데 그 지령은 어느 등록부안에 배치되였는가? 지령파일의 위치를 알아 내는 가장 손 쉬운 방법은 type지령을 사용하는것이다.

```
$ type ls
ls is /bin/ls
```

ls는 /bin등록부에 위치하고 있다. /bin은 PATH변수값의 구성요소이므로 체계는 그것을 쉽게 찾아내여 실행시키게 된다.

주해

만일 이 목록에서 지령을 찾을수 없다면 그 지령은 실행될수 없다는것을 의미한다. 이 경우에는 파일의 정확한 위치를 담고 있는 경로이름을 사용해야 한다. 그렇지 않으면 PATH변수를 변경시켜 지령파일이름을 가지고 있는 등록부를 그 목록에 첨부할수도 있다. 이 두가지 방법에 대해서는 후에 배우게 된다.

2.3 내부지령과 외부지령

우리는 이 모든 탐색작업들을 총체적으로 개괄하여 마치 어떤 체계 같은것이 있는듯이《체계》에 귀착시켜 놓았다. 이 모든 작업들을 실지로 수행하는 매개물은 **쉘**(shell)이다. 이것은 사용자가 체계에 가입하는 순간에 실행을 시작하는 특수한 지령이다. 쉘은 사용자가 입력하는 지령을 받아 들이며 자기의 PATH변수를 보고 그것이 어디에 위치하는가를 찾아 낸다.

ls는 /bin등록부(또는 /usr/bin)안에 독립적으로 존재하는 파일이므로 이것을 **외부지령**(external command)이라고 부른다. 대부분의 지령들은 본래 외부지령들이지만 그중에는 그 어디에서도 발견되지 않거나 설사 발견되였다고 해도 실행되지 않는것들이 일부 있다. 실례로 echo지령을 들수 있다.

```
$ type echo
echo is a shell builtin
```

echo를 입력하면 쉘은 그의 위치를 찾기 위하여 자기의 PATH를 보려고 하지 않는다(비록 /bin에서 그것을 찾아 낼수 있음에도 불구하고 PATH를 보지 않는다). 그대신 분리된 파일들로 저장되여 있지 않는 자기의 내장된 지령묶음으로부터 그것을 실행시킨다. echo가 포함되여 있는 이 내장된 지령들을 가리켜 **내부지령**(internal command)이라고 한다.

type지령은 쉘에 내장되여 있으며 사용자가 그것을 실행시킬수 있는가, 없는가 하는것은 그 사용자가 사용하는 쉘에 의존한다. C쉘의 일부 판본들에서는 그 지령이 동작하지 않는다. UNIX는 지령의 속성과 위치를 나타내는 여러가지 지령들을 가진다. 만일 type가 동작하지 않으면 같은 방법으로 whence나 which를 시도해 볼수도 있다.

UNIX는 다른것들과는 달리 리력을 가지고 있다. 많은 경우 지령은 외부지령으로 시작되였다가 그후 쉘에 흡수되여 그의 내부지령으로 되였다. 쉘은 이와 같은 내부지령들을 여러개 가지고 있다. 오늘날 일부 체계들은 기본적으로 사용되지 않음에도 불구하고 여전히 파일체계안에 그러한 외부지령(pwd와 같은)들을 보존하고 있다. 쉘은 /bin이나/usr/bin에 같은 이름을 가진 지령이 있다고 해도 최우선권은 무조건 그 이름을 가진 자기의 내부지령에 준다. 이것은 결국 그 외부지령이 전혀 실행되지 않는다는것을 의미한다.

echo의 경우를 보면 그 파일은 /bin에서 찾아 볼수 있지만 쉘은 언제나 내부의 echo를 외부보다 더 우선시하기때문에 이것은 좀처럼 실행되지 않는다. 이 지령 역시 실행시키는 방법이 있기는 하지만 쉘과 파일체계의 작업들에 대해 인식한 다음 해보기로 하자.

주해

일상적으로 사용되는 UNIX의 중요지령들은 /bin과 /usr/bin등록부들에 위치하고 있다. 도형출력을 보여 주는 지령들은 일반적으로 /usr/XllR6/bin이나 /usr/dt/bin에서 찾아 볼수 있다. 체계관리자가 사용하는 지령들은 /sbin과 /usr/sbin에서 찾아 볼수 있다.

2.4 지령구조

제1장에서 우리는 여러개의 단어를 가진 지령(who am i나 cat note1과 같은)들을 입력하였다. 또한 미누스부호가 섞여 있는 지령(ls -1과 같은)도 사용하였다. 이제는 전형적인 UNIX지령에 대해 구체적으로 학습할 때가 되였다. 이 지령구조에 대해 그림 2-1에서 보여 주고 있다.

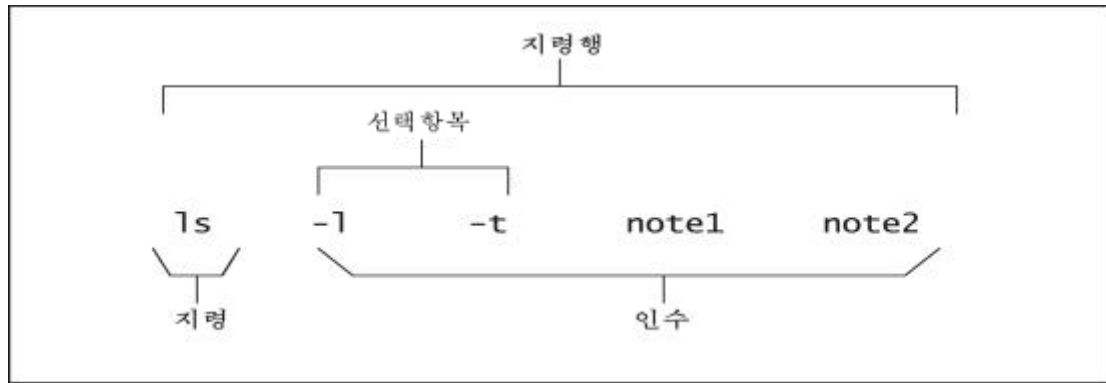



그림 2-1. UNIX지령구조

여기서는 전체 지령이 공간으로 분리된 5개의 **단어**(word)들로 이루어 져 있다. 첫번째 단어(ls)는 지령이고 기타 다른 단어들은 **인수**(argument)라고 부른다. 여기서 ls지령은 4개의 인수들을 가지고 있다. 이와 류사하게 am과 i는 who지령의 인수들이다. 지령은 종종 여러가지 형태의 인수를 받아 들이기때문에 다양한 방법으로 동작할수 있다.

지령들과 인수들은 여러개의 공간(space)과 타브(tab)들로 분리되여 체계가 그것들을 단어로 해석할수 있게 한다. 공간들과 타브들로 되여 있는 문자렬을 **공백**(whitespace)이라고 한다. 체계는 단어들을 분리시키는데 한개의 공백문자를 사용할것을 요구하지만 보통 아래의 지령에서와 같이 여러개를 허용하기도 한다.

```
ls -1 -t note1 note2
```

사용자는 이런 식으로 ls지령을 입력할수 있다. UNIX체계는 이 여러개의 련속공간들이나 타브들을 한개 공간으로 압축하는 특수한 기구를 가지고 있다. 건반에 있는 [Tab]건을 누르거나 만일 이 건이 없는 경우에는 [Ctrl-i]를 사용하여 타브문자를 만들어 낸다.

초학자들은 자주 지령과 인수사이에 공간을 주는것을 잊어 버린다. Windows에서는 DIR /P대신에 DIR/P라고 할수 있지만 UNIX에서는 서로 다른것으로 된다.

```
$ ls-1
ls-1: not found
```

체계는 -1을 선택항목으로 인식하지 못하고 ls-1을 한개의 지령으로 취급한다. 그러한 지령은 명백하게 존재하지 않는다.

주해

설사 who am i 외에도 whoami가 동작하는것을 발견했다고 해서 리론에 모순이 있다고 생각해서는 안된다. 이것은 whoami가 대부분의 UNIX체계들에서 볼수 있는 지령으로도 된다는것일뿐이다. 더우기 체계가 ls-1을 합법적인 지령처럼 취급한다는것을 느꼈다면 그것은 UNIX가 ls-1이라는 이름을 가진 별명(실제로는 ls -l을 실행하는)을 만들어 준것이다(17.4).

2.4.1 선택항목과 파일이름

실례와 그림 2-1에는 -부호로 시작되는 2개의 인수들이 있다. -1과 -t는 **선택항목**(option)이라고 부르는 특수한 인수들이다. 선택항목들은 자기들의 목록이 사전에 결정되였으므로 특수한 이름을 가지게 된다. 매개 지령은 고정된 선택항목묶음을 가지고 있다. 선택항목은 지령의 기본적인 작용을 변경시키므로 ls자체가 오직 파일이름만을 나타낸다면 -1과 -t선택항목들은 그 속성도 나타낸다.

선택항목들의 뒤에는 2개의 인수들(note1과 note2)이 놓이는데 여기서는 파일이름을 나타내고 있다. 일부 지령들은 파일들을 사용하지만 일부는 그렇지 않다. 사용자가 파일이름과 함께 어떤 지령을 실행시키면 지령은 그 파일을 열고 파일내용에 대한 모든 지시를 수행한다. 만일 전부 사용되면 파일이름은 보통 선택항목뒤에 놓이는 마지막인수일것이다(이것은 항상 그러한것은 아니며 일부 선택항목들은 파일이름을 자기의 인수로서 사용한다).

지령은 자기의 인수들이나 선택항목들과 함께 **지령행**(command line)이라고 하는 한개의 행에 입력된다. 이 행은 사용자가 [Enter]건을 눌러야 완성된다고 할수 있다. 완성된 행은 해석과 실행을 위한 입력으로서 쉘에 입력된다.

일부 지령들은 한개의 파일이름을 접수하고 어떤 지령들은 2개의 파일이름을 접수하며 또 일부는 무한정 접수하기도 한다(이것은 파일이름들을 수용할수 있는 체계용량에 따라 한정되게 된다). 여기에 파일이름들을 인수로 사용하는 지령들에 대한 몇가지 실례가 있다.

```
ls -1 -a -t chap01 chap02 chap03
cp chap01 chap02 progs
rm chap01 chap02
pine -f mail-feb            mail-feb는 -f의 인수이다
```

여기서 마지막지령은 다른것들과 약간 다르다. 여기서 mail-feb(파일이름)는 pine의 인수이기도 하지만 보다 정확하게는 -f선택항목의 인수이다. 이것은 -f뒤에 파일이름이 놓인다는것을 의미한다.

사용자가 잘못된 선택항목을 가지고 지령을 사용하는 경우 쉘은 지령의 위치는 정확히 알아 내지만 이때 그 지령은 선택항목이 틀린다는것을 발견하게 된다.

```
$ ls -z note
ls: ERROR: Illegal option -- z
usage: ls -1ACFLRabcdfgilmnopqrstux -W[sv] [files]
```

우의 통보문은 쉘이 아니라 지령에 의해 만들어 진것이다. ls는 많은 선택항목들(20개이상)을 가지고 있지만 -z는 가지고 있지 않다. 다른 많은 지령들도 사용자가 틀리게 사용할 때에는 정확한 문법과 선택항목들을 현시한다.

주의

일반적으로 선택항목앞에는 미누스기호를 붙임으로써 파일이름으로 구성된 다른 인수들과 구별한다. -기호와 선택항목문자들사이에 공백을 두어서는 안된다. 더우기 파일이름이 -로 시작되는 경우에는 대부분의 지령들이 제대로 작업하지 않을것이다.

2.4.2 선택항목들의 결합

UNIX에는 선택항목들의 사용을 규제하는 몇가지 규칙이 있다. -표식으로 시작된 선택항목은 보통 한개의 -표식과만 결합된다. 실례로 아래의 지령행은 3개의 선택항목들을 가지고 있다.

```
ls -1 -a -t
```

제7장에서 ls에 대해 구체적으로 언급되겠지만 -1선택항목이 거의 모든 파일속성들의 상세한 정보를 제공한다는데 대해서는 알고 있는것이 좋다. 만일 지령행에 이 선택항목만이 지정되였다면 그 파일들은 자모순서로 정돈되였을것이다(실지로는 ASCII순서이다). -t선택항목은 이 동작을 변경시켜 파일들을 변경된 시간에 따라 정돈한다. -a선택항목을 추가하면 체계의 숨은 파일들을 포함하게 된다. 이것들은 단순선택항

목들이며 UNIX는 이것들을 결합시킬수 있게 한다. 사용자는 ls지령을 다음과 같이 사용해도 동일한 출력을 얻게 된다.

```
ls -lat
```

또한 이것들을 임의로 결합하여 사용할수 있다

```
ls -tal                 결합순서가 언제나 중요한것은 아니지만
ls -atl                 때로는 중요하다
```

이 기능은 여러개의 선택항목들을 가지고 지령을 사용할 때 사용자의 타자부담을 줄이며 일정한 차이를 만든다. 쉘은 선택항목결합을 개개의 선택항목들로 분석(parse)한다.

일부 지령들은 우에서처럼 선택항목들을 결합하는것을 허용하지 않는다. 때로는 tar지령에서와 같이 선택항목뒤에 그 선택항목의 인수들이 놓이게 된다.

```
tar -cv -f /dev/fd0 -b 18 *
```

여기에는 4개의 선택항목들이 있으며 그중 2개(-f와 -b)는 특수한 선택항목들로서 자체의 인수들을 가지고 있다. 앞의 실례에서 우리는 pine이 -f선택항목을 동일한 방법으로 사용하는것을 보았다. 이것들을 **선택항목파라메터**(option parameter)라고 부르기로 하자. -f선택항목뒤에는 파일이름 /dev/fd0이 놓인다. -b선택항목은 블로크크기를 지정하며 여기서는 18이다. 이것이 바로 tar지령을 사용하여 현재등록부의 모든 파일들을 플로피디스크에 여벌복사하는 방법이다.

여기서 -c와 -v선택항목들을 결합하였는데 -f와 -b선택항목들도 결합시킬수 있다. 이 선택항목들의 파라메터들도 동일한 순서대로 배치한다면 가능하다. 실례로 다음의 결합이 원칙적인 결합이다.

```
tar -cvfb /dev/fd0 18 *
```

-f와 -b선택항목들뒤에는 그것들의 파라메터가 같은 순서로 놓이며 지령은 동작할것이다. 하지만 아래의 지령은 동작하지 않을것이다.

```
tar -cvfb 18 /dev/fd0 *
```

여기서 tar는 18을 -f의 파라메터로, /dev/fd0를 -b의 파라메터로 해석한다. 이것은 명백히 리치에 맞지 않는다.

주의

설사 파라메터들을 정확하게 줄 지어 세워 놓았다고 해도 모든 지령들이 선택항목결합을 허용하리라고 기대할수는 없다. 그러한 경우에는 선택항목들을 따로따로 지정하여야 한다.

2.4.3 례외와 가변성

모든 지령들이 선택항목과 인수들을 의무적으로 사용하지는 않는다. clear와 같은 지령들은 그 어떤 인수도 받아 들이지 않는다. who와 date지령들은 인수들이 지정될수도 있고 그렇지 않을수도 있다. ls지령은 더 가변적이며 아래와 같은 여러가지 방법으로 사용될수 있다.

- 인수가 전혀 없이(ls)
- 선택항목들과만(ls -1)
- 파일이름들과만(ls chap01 chap02)
- 선택항목과 파일이름들의 결합을 리용하여(ls -la chap01 chap02)

이 책에서는 선택항목을 사용하는 수많은 지령들을 설명하고 있다. 그것들중 절대다수가 지금까지 론의해 온 선택항목규칙들을 따른다. UNIX는 선택항목이 어떻게 되여야 하는가에 대한 자기 식의 생각을 가지고 있는 사람들에 의해 개발되였으므로 어떤 규정들을 공식화하든지간에 반드시 례외가 있게 된다.

실례들을 좀 더 보기로 하자. 때때로 선택항목이라는 말은 잘못된 이름으로 되군 한다. cut지령은 다음과 같은 선택항목들을 요구한다.

```
cut -d: -f7 passwd
```

이 지령은 파일 passwd로부터 7번째 마당을 추출해 낸다. -d선택항목 그자체는 선택적이지만(즉 그것을 사용하지 않을수도 있다.) -c나 -f선택항목들중 어느 하나는 지정되여야 한다. 리치상 이 선택항목들중 하나는 의무적이다.

여기서 pr지령은 파일 foo의 5페지로부터 인쇄를 시작하지만 -대신에 +를 선택항목의 접두사로 사용한다.

```
pr +5 foo
```

여기서 dd지령은 파일 boot.img를 플로피디스크에 복사한다. 이 지령은 두개의 열쇠단어 if와 of를 =와 함께 사용한다.

```
dd if=boot.img of=/dev/rdsk/f0q18dt
```

awk와 per1과 같은 일부 선진적인 UNIX지령들은 그자체가 프로그람작성언어들이다. 더우기 지령행에는 실지 인수로 취급되지 않는 일부 문자들이 있는데 실례로 |, <, >와 같은것들이다. 제8장에서는 이것들중 일부가 어떻게 지령앞에 놓일수 있는가를 보게 된다.

그러면 이 모든것으로부터 어떤 결론을 내릴수 있는가? 일반화된 지령문법을 지정한다는것은 정말로 불가능하다. 특정한 지령에 가장 적절한 문법은 UNIX안내서로부터 얻어 진다. 일부 지령들의 문법에 대해서는 이 책에 지정되여 있다. 하지만 이것으로 국한되여서는 안된다.

2.5 지령사용의 유연성

UNIX체계는 지령사용에서 어느 정도의 유연성을 제공해 주고 있다. 지령은 흔히 두가지이상의 방법으로 입력될수 있으며 사용자가 이것을 능숙하게 사용하면 건조작회수를 최소로 줄일수 있다. 이 절에서는 쉘이 지령사용에서 얼마나 관대한가 하는것에 대해 보게 된다.

2.5.1 지령들의 결합

지금까지는 지령들을 따로따로 실행시켰다. 매 지령은 처음것이 먼저 처리, 실행된후에야 입력되였다. UNIX는 사용자가 동일한 지령행에 두개이상의 지령을 지정할수 있게 한다. 매개 지령은 반두점(;)으로 구분되여야 한다.

```
who ; ls -1 note                ls는 who다음에 실행된다
```

이 지령들의 출력을 절환하는것을 배우게 되면 그것들을 괄호안에 함께 묶으려 할수도 있다.

```
( who ; ls -1 note ) > newlist
```

이 두 지령들의 결합된 출력은 파일 newlist로 보내여 진다. 여기서는 읽기 쉽게 하기 위하여 공백을 제공하였으나 아래와 같이 몇개의 건조작을 줄일수도 있다.

```
(who;ls -1 note)>newlist          선택항목주위에는 항상 공간이 있어야 한다
```

반두점(;)은 쉘이 리해하는 첫 특수문자이다. 지령행에 이 문자가 있게 되면 쉘은 그 량쪽에 있는 지령들을 따로따로 처리할 필요가 있다고 리해한다. ;을 비롯한 이러한 특수문자들을 가리켜 **메타문자**(metacharacter)라고 한다. ;과 같이 쉘에 특수한 의미로 인식되는 메타문자들이 여러개 있다. 쉘메타문자들에 대해서는 제8장에서 취급한다.

2.5.2 긴 지령행의 입력

일반적으로 지령은 건입력에 의하여 행에 입력된다. 말단의 폭이 80문자까지로 제한되여 있다고 해서 사용자가 한개 행에 하나 또는 여러개의 지령들을 80문자이상 초과하여 련속 입력하는것을 차단하지는 않는다. 지령은 다음행에로 넘쳐 나도 여전히 론리적으로는 한개의 행안에 있는것으로 된다.

일부 지령들은 긴 문법을 가지고 있으며 사용자는 종종 지령행을 여러개의 행으로 확장할 필요를 느끼게 된다. 그때 쉘은 지령행이 완성되지 않았다는것을 가리키기 위한 **2차프롬프트**(secondary prompt)를 보여 준다(보통 > 또는 ?). 아래에 echo지령이 일부 체계들에서 어떻게 동작하는가를 보여 준다.

```
$ echo "This is
> a three-line          2차프롬프트(>)가 나타난다
> text message"
This is                 인용부호가 닫기면 없어 진다
a three-line
text message
```

여기서 우리는 겹인용부호가 닫기기전에 [Enter]건을 두번 눌렀다. >기호는 인용부호가 닫길 때까지 [Enter]건을 누를 때마다 매번 나타난다. 한개 지령을 사용하는것외에도 사용자는 관흐름(8.8)으로 여러개의 지령들을 사용하는것도 필요하게 될것이다. 이 지령결합들을 개별적인 행으로 가르면 지령행을 리해하기가 더 쉬워 진다.

C쉘

우에서 언급한 echo지령은 C쉘에서는 동작하지 않는다. [Enter]를 누르기전에 역사선(\)을 입력하여야 한다. 더우기 C쉘에서는 2차프롬프트(?)가 다르며 1차프롬프트 역시 %로서 다르다.

```
% echo "This is\
? a three-line\
? message"
```

참고

[Enter]를 누른 다음에 >나 ?가 나타나는것은 일반적으로 적당한 인용부호나 괄호가 없기때문일것이다. 그것을 제공한후에도 문제가 지속되는 경우에는 [Ctrl-c] 또는 [Delete], [Ctrl-u]로 그 지령행을 제거할수 있다.

2.5.3 이전 지령이 끝날 때까지 기다릴것인가

UNIX는 전2중말단을 제공하는데 그것은 사용자가 임의의 시각에 지령을 입력할수 있고 체계가 그것을 원만히 해석할것이라는것을 의미한다. 사용자가 긴 프로그람을 실행시킬 때 $프롬프트는 프로그람 실행이 완성될 때까지 출현하지 않을것이다. 련속적인 지령들은 프롬프트를 기다리지 않고 입력될수 있

다. 이것들은 화면에 현시되지 않을수도 있다.

타자앞완충기(type-ahead buffer:림시적인 기억구역)가 있는데 이것은 모든 지령들을 저장하고 이전 지령이 다 실행된 다음에 그 지령들을 실행에 넘긴다. 만일 사용자가 정확히 입력하였다면 이전 프로그람의 출력이 화면을 차지하고 있다고 해도 전혀 기다릴 필요가 없다.

2.5.4 지령이 제대로 동작하지 않을수도 있다

UNIX는 많은 변종들을 가지고 있으므로 제작자들은 일정한 지령들을 전용화하여 AT&T가 정한대로가 아니라 자기들이 바라는대로 동작하도록 하였다. DOS에는 변종이 없으므로 그러한 문제들이 없다고 하지만 판본들사이에는 차이가 존재한다. DOS지령은 자주 UNIX지령과 같은 방식으로 판본에 따라 변화되며 따라서 사용자는 자기가 사용하는 DOS의 판번호를 알기 위하여 VER지령을 자주 사용하지 않으면 안된다. UNIX에서는 이러한 작업에 -r선택항목을 가진 uname을 사용한다.

```
$ uname -r                이 지령은 사용자에게
4.0                       기계이름도 보여 준다
```

이것은 실지 UNIX System V Release 4.0으로 확장된다. 때문에 만일 지령이 제대로 동작하지 못하면 다른 체계(BSD일수 있다.)나 또는 다른 판본(3.2일수도 있다.)에 속하게 될수 있다. 더 높은 판본들은 기능들을 버리는것이 아니라 거기에 더 보충한다.

Linux

uname -r지령은 핵심부의 판번호를 보여 준다.

```
$ uname -r
2.2.7
```

2.6 직결도움말(man)

사용자는 지령들이 대체로 어떻게 구성되는가 하는것과 지령들에서 보게 되는 가능한 가변성들에 대하여 일단 리해하기만 하면 UNIX문서를 볼수 있다. 오늘날 UNIX문서는 여러가지 형식으로 복잡하게 변화되였으며 여기서 가장 초기의것으로서 제일 중요한것은 man지령으로 현시되는 **man문서**(man documentation)이다. man은 UNIX체계에 대한 가장 완성되고(반드시 종합적인것은 아니지만) 믿음직한 안내자이다. 이 책에서는 후에 apropos, whatis, info도 사용하게 될것이지만 먼저 man에 대하여 설명하려고 한다.

man기능은 대체로 모든 체계들에 다 존재하지만 체계관리자가 디스크의 공간을 절약하기 위하여 일부러 설치하지 않을수도 있다. 문서는 인쇄에서도 역시 유용하다. C쉘의 안내페지를 보려면 csh(이 지령은 C쉘을 표현한다.)를 인수로 하여 man을 사용해야 한다.

```
man csh
```

csh지령에 관한 man페지전체가 화면에 현시된다. man은 첫 페지를 보여 주고는 림시 정지한다. 이것을 수행하기 위하여 man은 자기의 출력을 **페지화프로그람**(pager program)에 보내며 페지화프로그람은 파일내용을 한번에 한페지(한 화면)씩 보여 주게 된다. 다음페지를 보려면 건을 눌러야 한다(보통 [Spacebar] 또는 [Enter]). man을 완료하려면 q를 눌러야 한다.

실제로 페지화프로그람은 UNIX지령이며 man은 항상 특정한 페지화프로그람과 함께 사용되도록 미

리 구성된다. 현재 UNIX체계들은 다음의 페지화프로그람들을 사용하고 있다.

- more: 현재 광범하게 사용되고 있는 버클리의 페지화프로그람으로서 본래의 AT&T의 pg지령(지금은 사멸되였다.)에 비하여 아주 우월하다.
- less: 모든 Linux체계들에서 사용되고 있는 표준페지화프로그람일뿐아니라 모든 UNIX기계들에서도 유용하다. less는 vi편집기를 본 땄으며 vi의 항행 및 탐색기능을 거의다 도입한것으로 하여 more보다 더 강력하다.

man은 페지를 현시할 때 자기가 사용하고 있는 페지화프로그람에 대해서는 알려 주지 않는다. 사용자는 화면의 아래부분에 나타나는 통보문을 통하여 판단하여야 한다. 아래와 같은것이 보이면 more를 사용하고 있는것이다.

```
--More--(29%)            페지화프로그람은 more이다
```

more는 현시된 파일의 퍼센트를 보여 주고 있다. 그러나 두점(:)이 보인다면 pg를 사용하고 있는것이다(less의 일부 판본들도 프롬프트로서 :을 보여 준다). pg는 이미 사멸되였기때문에 우리는 그에 대해서는 고찰하지 않을것이다.

페지화프로그람에는 다음페지(일반적으로 이전 페지도 포함)를 보기 위한 건들이 정의되여 있다. 여기에는 프로그람을 완료하는 건도 있는데 이 건은 보통 q건이다. 일부 man페지들은 몇개의 페지들로 실행되는데 그러한 경우에 사용자는 문자렬들을 탐색할수 있다.

페지화프로그람이 동작할 때 사용자가 누르는 건들을 **내부지령**(internal command)이라고 할수 있다. 이 내부지령들은 쉘이 사용하는것들과 개념적으로 다르다. pine, vi, emacs와 같은 많은 UNIX지령들 역시 자기의 내부지령들을 가지고 있는데 그것들은 화면에 나타나지는 않아도 일정한 동작들을 수행한다. more와 less에 대해서는 제9장에서 취급하게 되는데 여기 표 2-1에서는 이 지령들에 대한 최소의 부분묶음을 보여 준다.

표 2-1. man의 페지화지령들

동 작	more	less
다음페지	[Spacebar] 또는 f	[Spacebar] 또는 f
10페지 앞으로 이행	10f	—
1000행 앞으로 이행	1000s	1000z(창문을 1000개 행으로 설정한다)
이전 페지	b	b
15페지 뒤로 이행	15b	—
1000행 뒤로 이행	—	1000w(창문을 1000개 행으로 설정한다)
첫 페지	—	p 또는 1G
행 300에 이행	—	300G
TCP를 탐색	/TCP	/TCP
탐색반복	n	n
마지막지령을 반복	.(점)	—
탈퇴	q	q
도움말현시	h	h

man문서는 때때로 범위가 아주 넓으며 탐색기능은 열쇠단어를 포함하고 있는 페지를 쉽게 찾아 낼수 있게 해준다. 실례로 사선(/)을 사용하여 단어 clobber가 있는 페지를 호출해 낼수 있다.

```
/clobber[Enter]
```

이때 /과 탐색문자렬이 화면에 나타나며 [Enter]를 누르면 그 페지에 가게 된다. 만일 그 페지가 사용자가 찾으려는 페지가 아니면 n을 눌러 탐색을 반복할수 있다. 페지화프로그람의 일부 판본들은 반전된 색으로 탐색항목을 강조하여 준다.

사용자들은 man을 한번 불러 내여 여러 지령들의 man페지들을 볼수 있다. 여기에 한가지 실례가 있다.

```
man cp mv rm
```

이 지령은 먼저 사용자에게 cp지령(UNIX가 파일복사에 사용하는 지령)에 대한 페지를 보여 준다. 다음지령을 보려면 q를 누른다. rm에 대한 페지에서 q를 누르면 man도 함께 완료될것이지만 중도에 끝내려면 체계의 새치기문자([Ctrl-c] 혹은 [Delete])를 사용한다.

man도 ls나 cat와 같은 UNIX지령이므로 사용자는 먼저 man지령 그자체가 어떻게 사용되는가를 알려고 할것이다. 동일한 지령을 사용하여 man의 문서를 볼수 있다.

```
man man
```

또한 우리는 사용자가 이 man페지로부터 자기의 페지도 선택할수 있다는것을 배울것이다. PAGER변수는 man이 사용하는 페지화프로그람을 조종한다. 만일 사용자가 이것을 less로 설정하면 man은 less를 자기의 페지화프로그람으로 사용한다. 하지만 Linux는 -P선택항목을 사용하여 서로 다른 페지화프로그람을 지정한다.

참고

more를 가지고 큰 man페지를 볼 때 사용자는 때때로 행이나 화면을 큰 묶음단위로 한번에 이동시키고 싶을 때가 있게 된다. 10f가 10페지를 뛰여 넘기는 하지만 사용자는 그 실제적인 지령을 처음에만 사용해야 할것이다. 그다음에는 more의 점(.)지령으로 그것을 반복할수 있다. 이 방법을 사용하면 조종이 훨씬 빨라 진다.

less에서는 이 기능을 리용할수 없지만 그대신 이와 류사한 단축기능을 가지고 있다. 200z로 200행 전진하였다면 z만을 눌러서 이 지령을 반복할수 있다. w와 z지령들앞에 수자가 붙게 되면 less는 창문크기를 재설정하는 기능을 가지게 된다.

주해

많은 UNIX체계들에서는 man이 사용하는 페지화프로그람이 PAGER변수안에서 유용하다. 이것을 알기 위해서는 echo $PAGER지령을 사용한다. man이 기정적인 페지화프로그람을 사용하는 경우 이것은 사용자에게 임의의 값을 보여 줄수도 있고 전혀 보여 주지 않을수도 있다. 대신 일부 체계들은 이 변수를 파일 /etc/default/man에 설정한다. 현재의 가입대화와 앞으로의 대화들을 위하여 PAGER변수값을 어떻게 변경시키는가에 대하여서는 후에 배우게 된다(17.3.1).

2.7 man문서

제작자들은 man문서를 서로 다르게 구성한다. 그러나 일반적으로 8개 부분으로 된 UNIX안내서를 보게 될것이다. 후에 보조적인 부분(1C, 1M, 3N 등과 같은)들이 보충되였으나 이 책에서는 그것들을 무시한다. SVR4와 Linux를 위한 기초적인 안내서부분들을 표 2-2에 보여 준다.

사용자가 사용하는 지령들의 대부분이 부분 1에 유용하므로 man은 부분 1에서부터 시작하여 안내서들을 탐색한다. 만일 한 부분에서 지령을 찾아 내지 못하면 man은 다른 부분에 지령이 있을수 있다고 해도 탐색을 더이상 하지 않는다. man이 대부분의 사용자지령들을 부분 1에 배치하므로 그것은 사용자가 1을 인수로 지정하지 않아도 된다는것을 의미한다.

```
man 1 cp                man cp와 같다
```

표 2-2. man문서의 구성

부분	제목(SVR4)	제목(Linux)
1	사용자프로그람	사용자프로그람
2	핵심부의 체계호출	핵심부의 체계호출
3	서고함수	서고함수
4	관리파일형식	특수파일(/dev안의)
5	기타	관리파일형식
6	유희	유희
7	특수파일(/dev안의)	마크로묶음과 규칙
8	관리지령	관리지령

우리는 -로 시작되지 않는 선택항목(1)을 사용하고 있으며 이것은 일반화된 지령문법으로부터의 첫 탈선으로 된다.

여기서 1은 중복되였다. 하지만 지령이 많은 부분들에서 발견될 때에는 선택항목으로서 부분번호를 추가적으로 사용해야 한다.

```
man 4 passwd          passwd는 부분 4에서도 나타난다
man -s4 passwd          solaris는 -s선택항목을 사용한다
```

Solaris기계들에서는 man -s4 passwd를 사용하여 부분을 지정해야 한다. 사용자는 또한 여러개의 부분번호들과 여러개의 지령들을 탐색하도록 지정할수 있다. 아래에 부분 2에서 mount문서를, 부분 4에서 passwd문서를 선택하는 방법을 준다.

```
man 2 mount 4 passwd          매 경우에 q를 사용하여 완료한다
```

이 지령은 부분 2로부터 mount지령의 페지들을 보여 주고 부분 4로부터 passwd지령이 사용하는 구성파일을 보여 준다. 이 파일 역시 동일한 이름 passwd를 가지지만 통과암호를 제외한 모든 사용자들의 상세한 정보들을 저장한다.

주해

man에 의해 수행되는 탐색의 기본적인 순서는 부분 1에서부터 시작된다. 때문에 지령(실례로 passwd지령)이 부분 1과 부분 4에 다같이 존재하며 사용자가 오직 부분 4에 있는 페지만을 보려고 한다면 부분번호를 인수로 하여 man을 사용해야 한다(man 4 passwd 또는 man -s4 passwd).

표 2-3. man페지머리부들

머리부	의 미
MAME	지령의 이름과 그의 기능을 보여 준다
SYNOPSIS	지령이 사용하는 인수와 선택항목들
DESCRIPTION	지령이 어떻게 사용되는가에 대한 상세한 내용
EXAMPLES	지령사용법의 복합적인 실례
FILES	지령이 사용하는 다른 체계파일들
SEE ALSO	완전한 리해를 위하여 관계되는 지령들의 man페지를 알려 준다
DIAGNOSTICS	지령이 오유통보를 발생하는 경우
BUGS	아직 바로 잡지 못한 지령안의 오유
AUTHOR(S)	그 지령의 작성자

man페지에 대한 리해

man페지구조는 여러 해동안 시종일관하게 유지되여 왔다. man페지는 여러개의 의무적이거나 선택적인 부분들로 나누이며 매 부분은 머리부를 가지고 있다. 표 2-3에 일반적인 머리부들에 대해 보여 준다.

매 지령이 모든 머리부를 다 가지고 있는것은 아니지만 대체로는 가지고 있다. 처음의 3개(NAME, SYNOPSIS, DESCRIPTION)는 일반적으로 모든 man페지들에서 다 볼수 있다. 그림 2-2에 일부 머리부를 가지는 압축된 man페지를 보여 준다.

```
User Command                                                  csh(1)

NAME
     csh - shell command interpreter with a C-like syntax

SYNOPSIS
     csh [ -bcefinstvVxX ] [ argument... ]

DESCRIPTION
     csh, the C shell, is a command interpreter with a syntax reminiscent
     of the C language.  It provides a number of  convenient features for
     interactive  use  that  are  not  available  with  the Bourne shell,
     including  filename  completion,  command aliasing,  history substi-
     tution, job control,  and a number of built-in commands. As with the
     Bourne shell, the C shell  provides  variable,  command and filename
     substitution.

OPTIONS
     -b       Force a "break" from option processing. Subsequent command
              line  arguments  are not  interpreted  as C shell options.
              This  allows  the  passing  of options to a script without
              confusion.   The   shell   does  not  run  set-user-Id  or
              set-group-ID scripts unless this option is present.

     -c       Execute  the  first  argument  (which  must  be  present).
              Remaining arguments are placed in argv,  the argument-list
              variable, and passed directly to csh.

FILES
     ~/.cshrc   Read at beginning of execution by each shell.
     ~/.login   Read by login shells after .cshrc at login.
     ~/.logout  Read by login shells at logout.

SEE ALSO
     bc(1),  echo(1),  login(1),   ls(1),   more(1),   ps(1),   sh(1),
     shell_builtins(1), tset(1B), which(1), df(1M), swap(1M), sysdef(1M),
     access(2),  exec(2),  fork(2),  pipe(2),  a.out(4),  environ(4)

DIAGNOSTICS
     You have stopped jobs.

          You attempted to exit the  C shell with stopped jobs under
          job control.  An immediate second attempt to exit will succeed,
          terminating the stopped jobs.

WARNINGS
     The use of setuid shell scripts is strongly discouraged.

BUGS
     As of this writing, the time built-in command does NOT compute the
     last  6 fields of output, rendering the output to erroneously report
     the value "0" for these fields.
```

그림 2-2. csh지령을 위한 man페지의 일부

안내서페지의 웃부분에는 지령이름과 괄호안의 수자를 보여 주는 제목이 있다. 이 지령은 안내서의 부분 1에 위치한다. 이 페지의 나머지부분을 더 자세히 보기로 하자.

- NAME(이름): 이 부분에서는 지령이름과 기능에 대한 간단한 설명을 보여 준다. 밀접하게 련관되여 있는 몇개의 지령들도 이 부분에서 함께 보여 준다. 실례로 일부 체계들에서 grep, egrep, fgrep는 아래와 같이 동일한 man페지를 공유한다.

```
grep, egrep, fgrep - print lines matching a pattern
```

- SYNOPSIS(개요): 문법 즉 그 지령과 함께 사용되는 선택항목들과 인수들을 보여 준다. 개요에는 모든 사용자들이 알아야 할 일정한 규칙이 있다. 하지만 안내서의 원본과 직결판본사이에는 이러한 규칙들이 약간 차이나게 된다. 원본은 굵은체, 사선체, 고정간격폰트를 사용하는데 이것을 대부분의 말단들은 조종할수 없다. SYNOPSIS를 구성하는데는 규칙이 있다.
  1. 현시된 안내서를 보면 지령이름과 그 선택항목들이 고정간격폰트로 표현되는것을 발견하게 될것이다. 이 폰트나 굵은체로 보여 준것은 무엇이든지 그대로 입력되여야 한다.
  2. 만일 지령인수를 꺾쇠괄호안에 넣었다면 이것은 선택적이다(인수를 줄수도 있고 주지 않을수도 있다). 그렇지 않은 경우에는 인수를 반드시 요구한다.
  3. 일반적으로 파일들은 사선체로 표현하며 이것은 거기에 실제적인 파일이름들이 삽입되여야 한다는것을 의미한다. 말단들에서는 이 파일이름들에 밑줄이 그어 져 있거나 색이 반전되여 있는것을 보게 될수도 있다.
  4. 생략부호(세 점들의 모임)는 앞의 단어가 련속적으로 반복된다는것을 의미한다.
  5. 이 구역들의 어떤 위치에서 |문자가 있으면 이것은 그 문자의 량쪽에 있는 선택항목들가운데서 오직 한개만을 사용할수 있다는것을 의미한다.
- DESCRIPTION(설명): 이것은 안내서페지의 가장 큰 부분이다. 여기서는 매 선택항목들에 대한 특별한 설명을 비롯하여 지령에 대한 구체적인 해설을 준다. 일부 체계들에서는 선택항목들자체가 자기의 개별적인 머리부를 가지고 있다. UNIX는 프로그람작성자들이 자기들을 위해 서술한것이기때문에 흔히 이 부분을 지나치게 가정하거나 오직 전문가만이 리해할수 있는 방법으로 정보를 표현하고 있다. 아래에 find지령에 대한 man페지의 일부를 보여 준다.

```
find searches the directory tree rooted at each given file
name by evaluating the given expression from left to right,
according to the rules of precedence, until the outcome is
known (the left hand side is false for and operations, true
for or), at which point find moves on to the next file name.
```

많은 사람들이 이것을 리해하지 못할수 있다. 그러나 이러한 상태들이 모두 일반적인것은 아니다.

- EXAMPLES(실례들): 때때로 복합선택항목들의 일부에 대한 사용을 강조하는 한두개의 실례가 있게 된다. 일부 경우에는 이것들이 모두 명백치 못하다는것을 발견하게 된다.
- FILES(파일들): 이것은 지령이 사용하는 모든 파일들을 보여 준다. 때때로 이 파일들의 내용도 보아야 할 필요가 있을수 있다.
- SEE ALSO(참고): 이것은 리해를 보다 충분히 하기 위해 체계의 다른 구성요소들을 참고하게 한다. 사용자는 man을 다시 사용하여 거기에 보여 준 지령들과 파일들에 대한 페지를 보아야 할수도 있다.

- DIAGNOSTICS(진단): 이것은 왜 그리고 언제 지령이 오유통보문을 발생시키는가에 대해 설명해 주고 있다. 때때로 오유코드들을 그 의미에 따라 렬거할수도 있다. 이 오유코드들은 쉘스크립트들에서 프로그람의 흐름을 조종하는데 사용될수 있다.
- BUGS(오유들): 이름이 암시해 주고 있는것처럼 이것은 사용자에게 그 어떤 문제를 야기시킬수 있는 프로그람의 오유들에 대해 알려 준다.
- AUTHOR(S)(저자): 일부 체계들 특히 Linux는 그 지령의 작성자들도 보여 준다. 여기서 사용자는 그들의 전자우편주소들도 볼수 있다.

앞으로 더 기회가 있겠지만 우리는 여기서 주요내용들을 먼저 취급하였다. 한두개의 man페지들을 가지고 있는 지령들은 사용하기 쉽지만 수십페지의 man페지를 가진 지령들은 그렇치 않다. 안내서들은 대체로 사용자가 체계를 충분히 파악한 다음에야 읽을수 있는 재료로 된다.

2.8 Texinfo문서(info)

Linux에서 tar지령의 man페지들을 탐색할 때 다음과 같은것을 볼수 있다.

```
$ man tar
No manual entry for tar
```

많은 체계들은 Texinfo문서(이것을 info문서라고 부르기로 하자.)도 지원하고 있으며 우와 같은 통보문을 보게 되면 info지령을 시도해 보아야 한다. 이 지령은 비록 처음에는 사용하기가 좀 어렵지만 초학자들에게 아주 적합하다. man과 같이 info문서는 완전무결할뿐아니라 사용자에게 많은 소개자료들도 제공해 준다.

이것도 지령이름과 함께 호출된다.

```
info tar
```

이제 사용자는 tar지령의 내용을 표로 서술하는 emacs식의 대면부를 보게 될것이다(그림 2-3).

```
        File: tar.info,  Node: Top,  Next: Introduction,  Prev: (dir), Up: (dir)

        * Menu:
Nodes   * Introduction::
        * Tutorial::
        * tar invocation::

        * operations::
        * Backups::
        * Choosing::
        * Date input formats::
        * Formats::
        * Media::
        * Index::

        -- The Detailed Node Listing --

        Introduction

        * Book Contents::              What this Book Contains
        * Definitions::                Some Definitions
        * What tar Does::              What `tar' Does
        * Naming tar Archives::        How `tar' Archives are Named
        --zz-Info: (tar.info.gz)Top, 248 lines --Top-- Subfile: tar.info-1.gz------
        Welcome to Info version 2.18. "C-h" for help, "m" for menu item.
```

그림 2-3. tar지령의 info페지들

info문서는 마디(node)로 구성되며 한개의 마디는 본문부분을 일정한 준위에서 표현한다. 여기에는 여러개의 준위가 있으며 사용자가 아래준위로 더깊이 내려 갈수록 취급은 더 발전한다. 다중준위문서들은 WWW에서 보게 되는 하이퍼본문(hypertext)문서들과 비슷하다.

이 체계에서 사용자는 먼저 마디들을 식별하여야 한다. 이것들은 행의 시작점에 별표로 표식되여 있다. 사용자는 이 행들중 임의의 행에 유표를 가져다 놓고 [Enter]를 누른다. 그러면 더 구체적인 내용들을 보여 주는 또 다른 폐지를 꺼내게 된다. 때때로 화면은 두개의 창문들로 나뉘여 보기 쉽게 해준다.

폐지안에서 사용자는 폐지를 넘기는 일반방법으로서 [PageUp]과 [PageDown]건들을 사용할수 있다. 사용자가 [Enter]건을 눌렀던 종전의 준위에로 돌아 가자면 u(up)를 누른다. 사용자은 또한 p(previous)와 n(next)을 사용하여 선형으로 이동할수 있다. info를 완료하려면 q를 사용한다. 이것들은 모두 사용자가 당분간 사용해야 할 지령들이다. 표 2-4에 기본적인 지령들에 대하여 보여 주고 있다.

표 2-4. info지령들

지 령	기 능
[Enter]	다음준위로 이행
u	이전 준위로 귀환
[Spacebar]	다음페지를 보기
[Delete]또는 [Backspace]	이전 페지를 보기
t	최상위마디를 보기
n	같은 준위의 다음마디를 보기
p	같은 준위의 이전 마디를 보기
q	info에서 탈퇴
h	도움말

다른 건들을 사용하지 않게 주의해야 하며 그렇치 않으면 도중에 정지될수도 있다. 만일 이런 경우가 생겨서 p와 u건들이 사용자의 요구대로 작용하지 못하게 되면 q로 info를 완료해야 한다. info읽기프로그람(reader)이 받아 들이는 건조작들의 완전한 목록을 보려면 h를 누른다. info문서는 구체적으로 잘 구성되여 있으므로 많은 Linux문서가 이 형식으로 이전되고 있다.

2.9 일감에 따르는 지령찾기(whatis, apropos)

man은 사용자에게 지령문법을 보여 준다. 그러나 특정한 환경에서 그 지령을 어떻게 적용하는가에 대해서는 보여 주지 않는다. 때때로 사용자는 그 지령이 무슨 일을 하는가에 대하여 알려고 하며 지령문법에는 들어 가지 않으려 한다. Whatis와 apropos지령이 이러한 목적에 더할나위없이 적합하다. 만일 사용자에게 이것들이 없게 되면 그와 대등한 선택항목을 가지고 있는 man을 사용할수 있다.

cp지령은 무슨 일을 하는가? whatis지령이 그에 대한 대답을 준다.

```
$ whatis cp
cp (1)            - copy files
```

이제는 사용자가 파일복사에 사용할 지령에 대하여 알게 되였다. 일단 필요한 지령을 식별하였다면 사용자는 man을 그 지령이름과 함께 사용하여 더 구체적인것들을 얻을수 있다. 괄호안의 수자는 그 지령이 부분 1에서 발견된다는것을 의미한다.

지령이 무엇을 하는가에 대하여 알고 싶은것이 첫째이고 실지로 그 일감을 수행하게 될 지령을 찾아내는것이 둘째이다. 만일 사용자가 주어 진 환경에서 사용할 지령에 대하여 전혀 모르고 있다면 한개이상의 열쇠단어들을 가지고 apropos지령을 사용해야 한다. 그러면 apropos는 사용자에게 그 열쇠단어를 포함하고 있는 모든 안내서부분들로부터 이름과 간단한 설명을 제공하여 준다.

```
$ apropos HTTP
Http (n)              - Client-side implementation of the HTTP/1.0 protocol.
b (8)                 - Apache HTTP server benchmarking tool
apachectl (8)         - Apache HTTP server control interface
httpd (8)             - Apache hypertext transfer protocol server
```

HTTP(WWW에서 사용되는 규약)를 취급하는 4개의 지령들이 있다. 출력은 또한 사용자에게 이 지령들을 발견하게 될 안내서부분들에 대해서도 말해 주고 있다.

whatis와 apropos는 다같이 여러개의 인수들과 함께 사용될수 있으나 만일 열쇠단어자체가 한개이상의 단어로 되여 있다면 인용부호로 그것들을 닫아 주어야 한다. 다음의것은 사용자가 정규식들에 대하여 더 배우려고 하는 경우에 쓸모가 있다.

```
$ apropos "regular expression"
re (3pm)              - Perl pragma to alter regular expression behavior
regexp (n)            - Match a regular expression against a string
regsub (n)            - Perform substitutions based on regular expression pattern
 matching
zgrep (1)             - Search possibly compressed files for a regular expression
```

첫 3개의 행들이 처음에는 흥미가 없을수도 있다. 하지만 이것을 통하여 압축된 파일안에서 표현을 탐색할수 있는 지령(zgrep)이 있다는것을 알게 된다.

참고

만일 사용자의 체계에 apropos가 없다면 man -k를 사용할수 있다. 또한 whatis대신에 man -f를 사용할수 있다.

Linux

대부분의 UNIX선택항목들이 -를 가진 한개의 문자를 사용하고 있다면 Linux는 2개의 -기호와 여러 문자단어를 사용하는 선택항목들도 제공한다. 실례로 ls -a에 동의어 ls --all을 추가로 제공한다. 이것은 사용자의 타자부담을 늘이는것으로 되지만 그 단어들은 의미가 명백하고 기억하기도 쉽다. 다시 말하여 -a보다 -all을 기억하기가 더 쉽다.

일부 Linux지령들은 man페지들을 가지지 않지만 그들중 거의 대다수가 모든 선택항목들의 함축된 목록을 현시하는 -help선택항목을 제공한다. 사용자는 아래의 지령을 사용하여 find의 선택항목을 찾을수 있다.

```
$ find --help
Usage: find [path...] [expression]
default path is the current directory; default expression is -print
expression may consist of:
operators (decreasing precedence; -and is implicit where no others are given):
   ( EXPR ) ! EXPR -not EXPR EXPR1 -a EXPR2 EXPR1 -and EXPR2
   EXPR1 -o EXPR2 EXPR1 -or EXPR2 EXPR1 , EXPR2
options (always true): -daystart -depth -follow -help
```

```
      -maxdepth LEVELS -mindepth LEVELS -mount -noleaf --version -xdev
   tests (N can be +N or -N or N): -amin N -anewer FILE -atime N -cmin N
      -cnewer FILE -ctime N -empty -false -fstype TYPE -gid N -group NAME
      -ilname PATTERN -iname PATTERN -inum N -ipath PATTERN -iregex PATTERN
      -links N -lname PATTERN -mmin N -mtime N -name PATTERN -newer FILE
      -nouser -nogroup -path PATTERN -perm [+-]MODE -regex PATTERN
      -size N[bckw] -true -type [bcdpfls] -uid N -used N -user NAME
      -xtype [bcdpfls]
   actions: -exec COMMAND ; -fprint FILE -fprint0 FILE -fprintf FILE FORMAT
      -ok COMMAND ; -print -print0 -printf FORMAT -prune -ls
```

Linux지령은 UNIX지령들보다 더 많은 선택항목을 제공하며 --help는 모든 선택항목들을 한 페지에 요약하여 보여 준다. 이 찾아보기기능이 선택항목사용법을 알려고 할 때 아주 쓸모가 있다는것을 깨닫게 되겠지만 사용자가 요구하는것을 재수집할수는 없다.

요 약

UNIX는 지령에 기초한 체계이다. UNIX지령들은 일반적으로 짧으며 소문자로 되여 있다. 이것들은 특정한 확장자를 가질 필요가 없다. UNIX는 대소문자를 구별한다.

type지령은 지령이 외부지령인가 내부지령인가를 보여 주며 외부지령의 위치도 알려 준다. 내부지령들은 쉘안에 내장되여 있으며 내부지령이 같은 이름의 외부지령보다 우선시된다.

쉘변수 PATH는 외부지령들의 위치를 찾아 내기 위한 등록부들의 탐색목록을 지정한다. PATH에서 사용되는 구분문자는 두점(:)이다. 모든 사용자들이 사용하는 체계지령들은 /bin과 /usr/bin등록부들에 위치하고 있다.

지령은 선택항목들과 인수들로 구성되며 대다수의 지령들은 파일들과 함께 사용된다. 선택항목은 지령의 기본적인 동작을 변경시키는데 대다수의 UNIX지령들은 몇개의 선택항목들을 가지고 있다. 선택항목들은 보통 -로 시작되며 지령과 인수들은 공백으로 분리되여야 한다. 선택항목들과 인수들을 가지고 있는 지령을 지령행이라고 한다.

선택항목들은 보통 한개의 -기호와 결합될수 있지만 일부 지령들은 자기자체의 파라메터들을 가지는 선택항목들을 사용한다. 일반적으로 그 파라메터들이 같은 순서로 배렬된다면 이 선택항목들도 결합될수 있다.

한개이상의 지령이 한행에 입력될수 있고 지령렬은 다음행으로 넘쳐 날수도 있다. 사용자는 이전의 지령이 완성되기를 기다리지 않고 또 다른 지령을 입력할수 있다. 만일 지령이 제대로 동작하지 않으면 uname -r를 리용하여 조작체계나 핵심의 판본을 검사할수 있다.

UNIX문서는 8개의 부분으로 구성되며 거의 모든 사용자지령들은 부분 1에서 찾아 볼수 있다. 지령이름과 함께 사용되는 man지령은 개요(문법), 설명, 지령이 사용하는 파일을 보여 준다. man은 본문을 한번에 한 페지씩 보여 주기 위하여 페지화프로그람 more 또는 less를 사용한다.

info는 여러 준위로 문서를 구성하며 지령사용과 관련된 정보를 man보다 더 상세히 제공한다. *로 시작되는 행에서 [Enter]를 누르면 그 화제에 대한 보다 상세한 내용을 제공한다.

Whatis(또는 man -f)는 지령에 대한 한 행분의 소개를 주며 apropos(또는 man -k)는 열쇠단어를 담고 있는 지령들의 목록을 제공한다.

Linux선택항목들은 앞에 두개의 -부호가 붙은 완전단어들이다. Linux지령은 보통 지령사용법을 신속히 볼수 있게 해주는 -help선택항목을 지원한다.

시험문제

1. 두점(:)을 입력하고 [Enter]를 누르면 어떻게 되겠는가?
2. UNIX지령들을 대문자로 실행시킬수 있는가?
3. 만일 지령이 .txt확장자를 가지고 있다면 그 지령이 실행될수 있는가?
4. 4문자이상의 길이를 가지는 지령들의 이름을 2개이상 불러 보시오.
5. PATH란 무엇인가?
6. PATH목록에서 등록부들을 분리시키는데 사용하는 문자는 무엇인가?
7. ls -1대신에 ls-1을 사용하는것이 가능한가?
8. 아래의 지령에 몇개의 선택항목들이 있는가?

```
ls -lut chap01 note3
```

9. 두개의 -부호가 붙은 선택항목(--all과 같이)들을 사용할수 있는것은 UNIX의 어느 변종인가?
10. 선택항목들은 반드시 -로 시작되여야 하는가?
11. 선택항목은 실지 선택적인가?
12. 지령과 그의 선택항목들, 인수들에 주어 진 이름은 무엇인가?
13. 어느 지령들이 실지로 프로그람작성언어들인가?
14. 조작체계의 판번호를 어떻게 찾아 낼수 있는가?
15. /bin과 /usr/bin등록부들이 왜 echo $PATH의 출력에서 항상 처음에 발견되게 되는가?
16. 다음지령을 입력하기전에 이전 지령이 끝나기를 기다려야 하는가?
17. 지령의 man페지란 무엇인가?
18. 여러 준위 또는 여러 마디들로 페지들을 구성하는 문서열람기는 무엇인가?
19. 일감을 수행할수 있는 지령의 이름을 전혀 모른다. 어떻게 하겠는가?

련습문제

1. 지령앞에 #를 입력하고 [Enter]를 누른다. 무엇을 보게 되는가? 어떻게 이 동작을 리용할수 있다고 생각하는가?
2. UNIX지령들과 Windows프로그람들사이의 3가지 주요차이점들에 대하여 말해 보시오.
3. 일반적으로 리용되는 UNIX지령들은 어디에 있는가?
4. 체계관리자가 리용하는 지령들은 어디에 위치하고 있는가?
5. /bin이나 /usr/bin에서는 cd지령을 찾을수 없을것이다. 그러면 이것은 어떻게 실행되는가?
6. echo, date, pwd, ls와 같은 지령들이 내부지령인지, 아니면 외부지령인지 어떻게 알아 낼수 있는가?

7. /bin등록부에서 echo지령을 찾아 내였다면 그것을 여전히 외부지령이라고 할수 있는가?

8. 선택항목도 인수인가? 인수와 선택항목은 어떻게 다른가?

9. 왜 사용자는 -로 시작된 파일이름을 가지지 말아야 하는가?

10. 건반 누르는 회수를 줄여서 다음의 지령을 실행해 보시오.

```
tar -t -v -f /dev/fd0
```

11. 지령 tar -x -v -f /dev/rct0 -b 20 *에서 아래와 같이 선택항목들을 결합시킬수 있는가?

```
tar -xvfb 20 /dev/rct0 *
```

12. +를 접두사로 하고 수자를 선택항목으로 사용하는 지령을 하나 불러 보시오.

13. who와 date지령들의 출력을 한개의 파일에 보관시키려면 어떻게 하여야 하는가?

14. 2차프롬프트는 어떻게 보이는가? 언제 나타나는가?

15. 다음의 man페지에서 SYNOPSIS부분의 |는 무엇을 가리키고 있는가?

```
/user/xpg4/bin/tail [ -f | -r ]
```

16. 아래의 SYNOPSIS부분에서 3개의 점은 무엇을 가리키는가?

```
ps [ -aAcdefjlLPy ] [ -g grplist ] [ -n namelist ] [[ -o format] ... ]
```

17. man이 사용하는 2개의 페지화프로그람은 무엇인가? 어느것이 우월하며 왜 그런가?

18. f를 반복하여 눌러서 ksh(korn쉘)의 man페지끝에 가닿았다. 페지화프로그람이 more라고 가정할 때 최소한의 건조작으로 어떻게 시작위치로 돌아 올수 있는가? less의 경우에는 어떻게 해야 하는가?

19. man이 사용하는 페지화프로그람에 대하여 짐작할수 없고 PAGER변수도 정의되지 않았다. 그러면 무엇을 시도하여 볼것인가?

20. /crontab[Enter]로 탐색하여 man페지에서 문자렬 crontab의 위치를 찾았다. 이 페지안에서 이 문자렬이 있는 다른 위치는 어떻게 찾아 내는가?

21. man에 대한 인수로서 부분번호를 사용하는것은 어느 때인가?

22. 체계에 apropos지령이 없다면 어떻게 하겠는가?

23. 파일체계의 태우기점을 조종하는 지령들을 렬거하기 위하여 지령 apropos mount point를 주었는데 그와 관련된 내용이 출력되지 않았다. 왜 그런가?

제 3 장. 범용편의프로그람

UNIX지령모임에 대한 지식을 얻기 위한 가장 좋은 방법은 체계의 일부 범용편의프로그람들을 알아보는것이다. 이 지령들은 여러가지 기능들을 가지고 있지만 크게 두가지 부류로 나눌수 있다. 일부 지령들은 체계상태 즉 현존사용자들과 날자, 사용자의 기계 및 말단이름들을 알려 준다. 다른 지령들은 체계에 가입하거나 수산기봉사를 제공하는 등 사용자의 작업을 직접적으로 도와 준다.

이 장에서 취급하는 매 지령들은 모두 유용한것들이며 결코 겉치레효과를 위하여 여기에 첨부시킨것이 아니다. 대부분이 다음장들, 특히 쉘프로그람작성부분에서도 사용된다. 사용자는 기계와 일상적으로 작업하는 모든 상황에서 이 지령들이 필요할것이다. 이 지령들은 사용하기가 간단하고 아주 적은 선택항목들을 가지며(stty를 제외하고는) 파일읽기쓰기를 거의나 하지 않는다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- passwd를 리용한 통과암호의 변경과 체계를 위한 통과암호구성규칙들에 대하여 배운다(3.1).
- who와 w를 리용하여 체계의 사용자들을 찾아 낸다(3.2).
- tty를 리용하여 사용자의 말단장치이름을 알아 낸다(3.3).
- lock로 사용자의 말단을 잠근다(3.4).
- stty를 사용하여 말단설정을 변경시킨다(3.5).
- script를 리용하여 모든 건누르기와 지령출력을 파일안에 보관한다(3.6).
- clear와 tput로 화면을 지우고 유표를 배치한다(3.7).
- uname으로 기계이름을 알아 낸다(3.8).
- date를 리용하여 체계날자를 여러가지 형식으로 현시한다(3.9).
- cal로 임의의 월과 년의 력서를 표시한다(3.10).
- calendar로 재생봉사를 발생시킨다(3.11).
- bc에서 수산기기능을 사용한다(3.12).

3.1 통과암호의 변경(passwd)

Windows환경(NT까지)에서 보안문제를 얼마 취급하지 않은것은 이 체계가 실지 보안환경을 그리 크게 마련해 줄수 없었기때문이였다. 하지만 UNIX는 가능하며 만일 사용자가 자기 파일들을 보호하는데 대해 실지로 관심이 있다면 그 누구도 사용자의 동의 없이는 사용자의 등록자리를 사용할수 없도록 할수 있다. 만일 사용자의 등록자리가 통과암호를 가지고 있지 않거나 혹은 다른 사람들이 이미 알고 있는 통과암호를 가지고 있는 경우에는 사용자는 즉시 그것을 변경시킬수 있다.

제1장에서 우리는 passwd지령을 사용하여 통과암호들을 변경시켰다. 이 지령의 동작은 체계에 깊이 의존하고 있다. 사용자의 통과암호를 설정하기 위하여서는 이 지령을 인수없이 사용한다.

```
$ passwd
(current) UNIX password: *******                    통과암호는 화면에 나타나지 않는다
```

```
New UNIX password: *******
Retype new UNIX password: *******
```

일반사용자가 이 지령을 호출하면 낡은 통과암호를 물어 보고 그후에 새 통과암호를 두번 요구한다. 만일 모든것이 순조롭게 진행되면 새 통과암호가 체계에 등록되며 프롬프트가 돌아 오게 된다.

그것들이 구성된 방법에 따라 체계들은 사용자가 통과암호로 입력한 문자렬에서 일정한 검사를 진행한다. 그것들은 사용자가 기억하기 쉬운 통과암호들을 배렬하려는것을 불허할수도 있고 나쁜 통과암호를 선택하지 말도록 권고할수도 있다. 아래에 아주 보편적인 통보문들이 있다.

```
UX:passwd: ERROR: Passwords must differ by at least 3 positions
passwd(SYSTEM): The first 6 characters of the password must contain at least
two alphabetic characters and at least numeric or special character.
passwd(SYSTEM): Password too short - must be at least 6 characters.
BAD PASSWORD: it does not contain enough DIFFERENT characters
BAD PASSWORD: it is based on a dictionary word
BAD PASSWORD: is too similar to the old one
```

이 통보문들은 사용자가 자기가 좋아 하는 임의의 통과암호를 마음대로 선택할수 없다는데 대해 시사해 주고 있다. 아래에 사용자가 통과암호조작에서 지켜야 할 일부 규칙들이 있다.

- 낡은 통과암호와 류사한 통과암호를 선택하지 말아야 한다.
- 친구들이나 친척들의 이름, 애완용동물들의 이름 등과 같은 통속적인 이름들을 통과암호로 사용하지 말아야 한다. 체계는 자기의 등록부를 검사해 보고 짐작으로 알아 맞힐수 있는 통과암호들은 제거해 버린다.
- 자모와 수자들을 섞어 사용해야 한다. UNIX체계들은 자모로만 되여 있거나 순수 수자로만 되여 있는 통과암호들을 허용하지 않는다.
- 통과암호가 다른 사람들이 짐작할수 없는 무의미한 뜻으로 되여 있는가에 대해 확인해 보아야 한다.
- 쉽게 접근할수 있는 문서안에 통과암호를 써넣지 말아야 한다.
- 통과암호를 정기적으로 변경시켜야 한다.

통과암호를 입력하게 되면 체계는 그 문자렬을 **부호화**(encryption)한다. 부호화는 자유문자들처럼 보이는 문자렬을 만들어 내며 UNIX는 후에 이것을 사용하여 통과암호의 정확성을 결정한다. 이 부호는 /etc등록부안에 있는 shadow라는 이름을 가진 파일에 보관된다. 어떤 사용자가 그 파일에서 이 부호를 볼수 있다고 해도 그 부호로부터 거꾸로 본래의 통과암호문자렬을 도출해 내지는 못한다.

통과암호들의 사용과 제거를 관리하는 정교한 규정들이 있다. 그리고 이 지령을 체계관리자가 사용할 때에는 좀 다르게 동작한다. 이 지령에 대해서는 제22장에서 다시 취급되게 된다.

주의

체계는 통과암호가 없이도 설정될수 있다. 이 경우에는 passwd지령을 실행시켜 즉시 바로 잡아야 한다. 자기의 통과암호를 다른 사람들이 알게 되였을 때에도 역시 통과암호를 변경시켜야 한다.

3.2 사용자알아보기(who, w)

UNIX는 체계에 가입한 모든 사용자들의 등록자리를 가지고 있다. 통보문들을 우편으로 보내거나 대화기간(11.2)을 설정할수 있도록 그들의 가입이름을 알고 있는것이 좋은 경우도 있다. 사용자들에 대한 정보목록을 현시하는 2개의 지령들이 있는데 그것들은 who와 w이다. who는 3렬로 된 간단한 출력을 만든다.

```
$ who
root         console      Jan 30 10:32
romeo        tty01        Jan 30 14:09
andrew       tty02        Jan 30 14:15
juliet       pts/4        Jan 30 13:17
```

첫번째 렬은 현재체계상에서 작업하고 있는 4명의 사용자들의 사용자ID를 보여 주고 있다. 두번째 렬은 각각의 말단장치이름들을 보여 주고 있다. romeo는 자기의 말단과 관련한 이름 tty01을 가지고 있다. 세번째 렬은 가입한 날자와 시간을 보여 주고 있다.

머리부정보를 현시하지 않는것이 UNIX지령들의 일반적인 특징이지만 이 지령은 머리부선택항목(-H)을 가지고 있다. 이 선택항목은 렬머리부를 현시하며 -u선택항목과 결합되면 보다 구체적인 목록을 제공한다.

```
$ who -Hu
NAME         LINE    TIME           IDLE    PID     COMMENTS
romo         tty01   Jan 30 14:09    .      30
andew   tty02   Jan 30 14:15    0:40    31
```

보는바와 같이 2명의 사용자들이 체계에서 탈퇴하였다. 첫 3개 렬들은 종전과 같지만 4번째 렬(IDLE)이 바로 흥미 있는것이다. romeo의 반대켠에 있는 .은 지령이 호출되기 마지막 1분전에 동작이 일어 났다는것을 보여 주고 있다. andrew는 마지막 40분동안에 아무 일도 하지 않은것처럼 보인다. 당분간 PID속성은 무시하기로 한다. 제10장에서 이에 대하여 다시 언급한다.

참고

who am i나 whoami로 자기의 사용자이름을 알아 볼수 있다. 이에 대해서는 이미 제1장에서 설명하였다.

주해

who출력에서 보게 되는 말단이름들은 실지로는 그 장치들을 표현하는 특정한 파일들이다. 이 파일들은 /dev에 위치한다. 실례로 파일 tty01은 /dev등록부에서 찾아 볼수 있다. pts/4는 /dev아래의 pts등록부아래에 있는 4라고 이름 지은 파일이다.

w지령은 사용자들의 동작에 대한 보다 상세한 출력을 만들며 추가적으로 체계에 대한 많은 세부정보들을 현시한다.

```
$ w
  2:40pm  up  1:37,  3 users,  load average: 0.00, 0.00, 0.00
```

```
USER        TTY   FROM      LOGIN@    IDLE   JCPU    PCPU    WHAT
romeo       tty1            1:24pm    6.00s  11.20s  11.01s  vi ux3rd03
andrew      tty2            2:20pm    6:35    0.68s   0.27s  ispell ux3rd17
juliet      tty4            2:42pm    1:10m   0.13s   0.13s  -bash
```

먼저 출력의 첫행을 보기로 하자. 이 지령은 오후 2시 40분에 실행되였으며 체계는 3명의 사용자를 가지고 있다. 체계자체는 1시간 37분동안 실행되여 왔다. 지난 1분, 5분, 15분동안의 체계평균부하는 무시할만한 정도이다.

마지막 3개 렬을 제외한 나머지출력들은 who에서와 같다. JCPU아래에 보여 준 출력은 그 말단에서 모든 프로쎄스들이 소비하는 CPU의 전체 시간이다. 사용자가 현재 실행시키고 있는 지령은 마지막렬에 보여 주고 있다. PCPU는 그 프로쎄스가 소비한 시간을 보여 준다. UNIX의 다중과제처리특성은 한명의 사용자가 한번에 한가지이상의 일감을 실행할수 있도록 허용하므로 JCPU와 PCPU시간들은 흔히 다르게 된다.

체계관리자는 who와 w를 정기적으로 사용하여 말단들이 제대로 리용되고 있는가를 감시한다. 이 두 지령이 다 관리작업에 매우 쓸모 있는 다른 여러개의 선택항목들을 제공한다. 수백명의 사용자들을 가진 대규모체계들에서는 출력이 너무 길어서 효과적으로 사용할수 없다. 한명의 사용자만을 위한 세부정보를 얻자면 w를 그 사용자이름과 함께 사용해야 한다.

참고

users지령을 사용하면 모든 사용자들에 대한 한행분의 목록을 얻을수 있다. users는 사용자들이나 체계에 대한 상세한 정보는 보여 주지 않는다. 이 지령은 모든 체계들에 다 유용한 것은 아니다.

3.3 말단알아보기(tty)

UNIX는 말단들을 파일로서 취급한다는데로부터 사용자가 현재 리용하고 있는 말단의 파일이름을 알려 주는 지령을 가지고 있다. 그것이 바로 tty(teletype)지령이다. 이 지령은 단순하며 아무 인수도 요구하지 않는다.

```
$ tty
/dev/term/2
```

말단파일이름은 term등록부에 존재하는 2이다(2라고 이름 지은 파일). 이 term등록부는 /dev등록부 아래에 위치하고 있다. 만일 다음번에 사용자가 다른 말단으로부터 체계에 가입하면 그의 말단장치이름이 달라 질것이다.

사용자는 쉘스크립트안에 tty를 사용하여 스크립트의 동작을 그것이 호출된 말단에 따라 조종할수 있다. 만일 프로그람이 어느 한 지정된 말단으로부터만 실행되여야 한다면 스크립트는 tty를 사용하여 그것을 결정하게 될것이다.

3.4 말단의 잠그기(lock)

때때로 사용자는 자기의 말단을 한동안 떠나 있어야 하겠지만 배경에서 일감이 실행되고 있으므로 체계에서 탈퇴하지 못할 때가 있게 된다. 많은 UNIX체계들은 나쁜 목적을 품은 사람들이 체계에 접근하

는것을 막기 위해 사용자의 말단을 잠그도록 되여 있다. 이것을 lock지령으로 수행한다. 사용자가 말단을 잠그려고 할 때에는 통과암호를 입력하여야 한다.

```
$ lock
Password: ********            화면에 통과암호가 나타나지 않는다
Re-enter password: ********
terminal locked by romeo 0 minutes ago
```

$프롬프트는 나타나지 않으며 체계는 이 상태에서 30분동안 잠그어 져 있게 될것이다. 만일 그 시간내에 사용자가 돌아 오지 않으면 사용자를 체계에서 탈퇴시키게 된다. 이러한 체계탈퇴가 진행되기전에 사용자는 어느 때든지 같은 통과암호를 다시 입력하여 간단히 말단을 열수 있다.

```
*********
$ _
```

lock가 요구하는 통과암호는 passwd지령으로 설정한 통과암호와 다르다. 사용자는 선택항목을 사용하여 말단을 잠그는 시간을 설정할수 있다(하지만 60분을 초과하지 말아야 한다).

```
lock -45                45분동안 잠그어 둔다
```

이 파라메터들은 가변적이며 일부 체계들에서는 파일 /etc/default/lock의 두가지 설정에 의하여 조종된다.

주해

/etc등록부안에 있는 임의의 파일은 뿌리를 사용자ID로 사용하는 체계관리자에 의해서만 변경될수 있다.

3.5 말단특성을 설정하기(stty)

말단은 사용자가 체계와 통신하는 장치이다. 사용자의 선택에 따라 여러가지 말단들이 각이하게 구성된다. 또한 사용자의 말단이 사용자가 의도하는대로 동작하지 않을수도 있다. [Enter]건이 동작하지 않을수도 있고 또 [Ctrl-c]로 무시(aborting)할수 없을수도 있다. 또한 사용자가 서로 다른 특성을 가진 새 말단을 련결할 때 그 파라메터들을 설정할 필요가 있을수도 있다. 이러한것들은 stty지령으로 수행할수 있다.

stty는 대단히 많은 열쇠단어(서로 다르게 보이는 선택항목들)를 사용하지만 우리는 그중 일부만을 고찰해 보려고 한다. -a(all)선택항목은 현재의 설정을 현시한다. 아래에 출력을 정돈하여 보여 준다.

```
$ stty -a
speed 38400 baud; rows = 25; columns = 80; ypixels = 0; xpixels = 0;
intr = DEL; quit = ^\; erase = ^h; kill = ^u;
eof = ^d; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^q; stop = ^s; susp = ^z; dsusp = ^y;
isig icanon -xcase echo echoe echok -echonl -noflsh
```

출력은 말단의 보속도(baud rate)를 보여 주며 우의 경우에는 38,400이다. 또한 제1장에서 설명된 많은 파라메터들도 보여 주고 있다. 이 체계에서는 [Delete]건([Ctrl-c]가 아니라)으로 프로그람을 중단

한다. 지우기문자는 [Ctrl-h]이고 제거문자는 [Ctrl-u]이다.

특별히 중요한것은 eof(파일끝)문자인데 여기서는 [Ctrl-d]로 설정되여 있다. 제1장에서는 이 건을 cat지령과 함께 사용하였다. 건반으로부터 입력을 받아 들이는 지령들에서는 이 건이 입력의 끝을 의미한다. 이 장에서는 이 건을 bc지령과 함께 사용하게 된다.

4번째 행다음에는 일련의 열쇠단어들을 보게 되는데 그중 일부는 앞에 -가 붙어 있다. 선택항목이 -를 가지고 있지 않다면 이것은 그 선택항목이 켜져 있다는것을 의미한다. 사용자는 stty지령을 사용하여 이 선택항목들을 설정하거나 해체할수 있다.

후진건이 문자를 지우는가(echoe)

사용자가 여러개의 말단들에서 작업할 때 문자들에 대한 후진건의 작용이 어떤 때에는 문자들을 시야에서 사라지게 하고 또 어떤 때에는 그렇게 하지 않는것을 발견하게 된다. 이것은 열쇠단어 echoe에 의해 결정된다. 여기서 그것이 설정되여 있으면(-가 붙어 있지 않으면) 후진건은 화면에서 그 문자를 제거한다.

이 설정을 반전시키려면 동일한 열쇠단어를 사용할수 있다. 여기서 echoe열쇠단어의 앞에 -를 붙여 주어야 한다.

```
stty -echoe
```

현재 후진건은 문자를 시야에서 제거하지 않고 있다. 이 설정은 일부 체계들에서는 동작하지 않는다.

쉘스크립트를 통한 통과암호의 입력(echo)

사용자들은 통과암호와 같은 문자렬이 화면에 나타나는것을 좋아 하지 않는다. echo의 설정은 바로 이러한 문자렬을 쉘프로그람들이 접수하도록 조종되여야 한다. 기정적으로 선택항목은 설정되지만 사용자는 아래의 방법으로 그것을 해제할수 있다.

```
stty -echo
```

이러한 설정으로 하여 건입력은 현시되지 않게 된다. 입력이 완성된 다음에는 stty echo를 사용하여 그것을 해제해야 한다. 그것은 다시는 화면에 현시되지 않지만 차후의 모든 입력에 대해 확인한다.

새치기건의 변경(intr)

stty는 또한 일부 건들의 기능도 설정한다. 실례로 사용자가 새치기건으로서 [Delete]대신에 [Ctrl-c]를 사용하려 하는 경우에는 아래의 지령을 사용하여야 할것이다.

```
stty intr \^c            ^와 c
```

여기서 열쇠단어 intr뒤에는 공백이 놓이고 뒤이어 \과 ^가 놓이며 끝으로 문자 c가 놓인다. stty는 이런 방법으로 체계에 새치기문자가 [Ctrl-c]라고 가리켜 준다.

조종문자들을 파일에 삽입할 때에는 문자앞에 부호가 있는것을 보게 될것이다. 실례로 [Ctrl-l]은 ^l(또는 ^L)로 보인다. 하지만 이것은 실제로는 말단에서 2개의 홈을 차지하고 있는 한 문자이다.

사용자는 stty의 인수로서 [Ctrl-c]를 누를수 없으므로(이것이 프로그람을 완료시킬수 있기때문에) ^기호가 사용되며 그앞에 \가 놓인다.

파일끝건의 변경(eof)

cat로 파일을 만들 때 입력을 완료하기 위해서는 [Ctrl-d]를 사용한다. eof문자도 선택될수 있다. eof문자로서 [Ctrl-d]대신에 [Ctrl-a]를 사용할수 있다.

```
stty eof \^a
```

이제는 [Ctrl-a]가 지령들에 대한 입력을 완료할것이다. cat와 bc지령을 이러한 방법으로 동작하도록 만들수 있다.

다른 모든것들이 실패하는 경우(sane)

stty는 말단특성에 값을 설정하기 위한 또 하나의 인수를 제공한다. 단어 sane을 한개의 인수로 하여 stty지령을 사용하여 보자.

```
stty sane                말단에 본래의 상태를 회복한다
```

다른 선택항목들이 더 있으나 여기서는 지나치게 많은 설정들을 취급하는것은 그만 두기로 한다.

3.6 대화기록(script)

script지령은 대체로 UNIX사용자들에게 잘 알려 져 있지 않은 지령으로서 사용자의 가입대화를 파일안에 기록하게 한다. 모든 지령들은 자기들의 출력과 오유통보문들을 나중에 보기 위해 파일에 저장된다. 만일 사용자가 몇가지 중요한 작업을 하고 있으면서 자기의 모든 작업내용을 기록하여 보존하려고 한다면 체계가입후 즉시 이 지령을 호출하여야 한다.

```
$ script
Script started, file is typescript
$ _
```

프롬프트가 귀환되며 사용자가 여기에 입력하는 모든 건조작들이 typescript파일에 기록된다. 기록이 다 끝나게 되면 exit를 입력하여 대화를 완료할수 있다.

```
$ exit            혹은 [Ctrl-d]를 사용한다
Script done, file is typescript
$ _
```

이제는 cat지령으로 이 파일을 볼수 있다. script는 이전의 typescript가 있으면 그 파일에 덧쓰기한다. 그 파일에 추가하거나 다른 기록파일을 사용하려고 한다면 아래의 선택항목을 참고할수 있다.

```
script -a                현존파일 typescript에 동작들을 추가한다
script logfile           파일 logfile에 동작들을 기록한다
```

제대로 기록되지 않는 동작들이 일부 있는데 실례로 전화면방식으로 동작하는 지령(vi, emacs, pine과 같이)들을 들수 있다.

3.7 화면지우기(clear, tput)

거의 모든 UNIX체계들에서는 화면을 지우기 위하여 2개의 지령 즉 clear와 tput를 사용한다. 첫번째 지령은 인수없이 사용한다.

```
clear            프롬프트가 보이지 않는다
```

화면은 지워 지고 유표는 화면의 왼쪽웃구석에 위치한다. 두번째 지령은 clear인수와 함께 사용한다.

```
tput clear            역시 화면을 지운다
```

tput는 유표를 특정한 위치에 놓이게 할수도 있고 본문을 강조하는데 사용될수도 있다. 그의 대부분의 인수들은 -부호로 시작되지 않는다. 다음과 같이 cup인수를 사용하면 유표가 10번째 행 20번째 렬에 위치하게 할수 있다.

```
tput cup 10 20
```

쉘스크립트에서 이 지령다음에 echo지령을 제시하게 되면 본문이 그 위치에 현시되는것을 볼수 있다. 쉘프로그람을 작성할 때 우리는 어느 한 련습문제에서 이 기능을 사용하여 볼수 있을것이다.

smso와 rmso인수들을 사용하면 본문을 굵은 문자로 만들수 있다. 아래의 실례에서 앞의것은 굵은 문자를 설정하고 뒤의것은 그것을 해제한다. 이것들은 보통 echo와 함께 사용된다.

```
tput smso               강조가 시작된다
echo Come to the Web
tput rmso               강조가 끝난다
```

나중에 우리는 한개의 지령에서 이것들을 결합하는 쉘의 특수한 기능을 사용하게 될것이다.

3.8 기계의 이름알아보기(uname)

만일 사용자의 기계가 망에 접속되여 있다면 그 기계는 틀림없이 이름을 가지고 있다. 망이 인터네트에 련결되여 있다면 이름은 그 기계의 령역이름의 한 부분을 형성한다(hillftp.planets.com과 같은). -n선택항목을 가진 uname지령은 망에서의 기계이름을 사용자에게 알려 준다.

```
$ uname -n
hillftp                 령역이름의 첫 단어
```

많은 UNIX망편의프로그람들이 기계이름을 인수로 사용한다. 원격기계로부터 파일들을 복사하려면 ftp지령과 함께 그것을 지정해야 한다. 동일한 uname지령이 -r와 함께 사용될 때에는 조작체계의 판번호를 보여 준다.

3.9 체계날자의 현시(date)

UNIX체계는 내부시계를 가지고 있다. 체계가 정지되면 축전지가 시계를 계속 돌려 준다. 실제로 이 시계는 1970년 1월 1일로부터 현재까지의 초단위값들을 보관한다. 32bit계수기가 이 값을 보관하며 2038년의 어느 때에 가서는 값이 넘쳐 나게 될것이다.

date지령을 사용하면 현재의 날자를 현시할수 있으며 가장 가까운 초에 대한 날자와 시간을 보여준다.

```
$ date
Sat Feb 12 23:10:34 EST 2000
```

이 지령은 적합한 형식지정자들을 인수로 하여 사용할수도 있다. 매 형식은 +부호가 앞에 붙으며 그 뒤에 %연산자와 형식를 서술하는 한개의 문자가 놓인다. 실례로 +%m형식을 사용하여 월(수자 또는 이름)을 현시할수 있다.

```
$ date +%m
02

$ date +%h
Feb
```

뿐만아니라 이것들을 한개의 지령으로 결합할수도 있다.

```
$ date +"%h (%m)"
Feb (02)
```

다른 형식지정자들도 있다.

d - 월의 날자(1부터 31까지)
y - 년의 마지막 두 수자
H,M,S - 각각 시간, 분, 초

여러개의 형식지정자들을 사용할 때에는(우의 실례에서와 같이) 그것들을 인용부호안에 넣어야 하며 인용부호를 열기전에 한개의 +부호를 사용해야 한다.

주해

일반사용자들은 날자를 변경시킬수 없지만 체계관리자는 동일한 지령을 다른 문법으로 사용하여 체계날자를 설정한다. 이에 대해서는 제22장에서 설명한다.

3.10 력서프로그람(cal)

cal은 사용자가 어느 때든지 호출하여 임의의 월이나 1년분의 력서를 볼수 있는 편리한 도구이다. 12월 력서를 보려면 달을 표현하는 세 문자략어를 인수로 주어야 한다.

```
$ cal dec
   December 2000
```

```
Su  Mo  Tu  We  Th  Fr  Sa
                     1   2
 3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25  26  27  28  29  30
31
```

cal은 또한 1년분의 력서를 현시할수 있다. 이때에는 년을 인수로 사용한다.

```
cal 2000
```

한 페지의 화면에 1년분 력서를 다 담을수는 없다. 그것은 너무 빨리 흘러 가 버리기때문에 미처 [Ctrl-s]를 사용하여 멈춰 세울수 없다. man에서와 같은 같은 방법으로 cal을 멈춰 세우려면 페지화프로그람(more 또는 less)을 함께 사용하여야 한다.

```
cal 2000 | more                    또는 less를 사용한다
```

|부호는 두 지령을 관흐름(pipeline)으로 련결하며 more는 cal지령으로부터 입력을 얻는다. 이제는 [Spacebar]를 눌러서 화면을 전진시킨다.

류사한 많은 편의프로그람들과는 달리 cal지령은 매우 정확하며 또한 1752년에 진행된 윤년조정도 고려하고 있다.

3.11 효과적인 재생기구(calendar)

calendar지령은 사용자에게 쓸모 있는 재생기구(reminder mechanism)를 제공한다. 이 지령은 현재등록부안에 있는 calendar라는 이름을 가진 파일에서 오늘이나 래일을 표현하는 날자를 포함하고 있는 행들을 탐색한다. 그다음 정합되는 행들을 그 말단에 현시한다. 대표적인 calendar파일은 다음과 같은것들을 보여 준다.

```
$ cat calendar
Mar 23, 2000 the target for this month is $3.2 million
The AGM is scheduled for March 24
Board meeting on 24th March, 2000 at 10 a.m.
On march 24 -- principals visiting us for discussion.
On 03/27/00 -- meeting with all sales and marketing people
alf-yearly results should be published by Mar 25, 2000
unch with the Chairman on Mar 26
```

파일은 일련의 통보문들로 구성되며 매 통보문은 몇가지 형식으로 입력된 일정한 날자를 포함하고 있다. calendar가 이 형식들을 얼마나 잘 리해하는가를 시험하기 위하여 오늘날자를 확인한 다음 먼저 이 지령을 실행시켜 보자.

```
$ date
Thu Mar 23 16:09:35 EST 2000
$ calendar
```

```
Mar 23, 2000 the target for this month is $3.2 million
The AGM is scheduled for March 24
On march 24 -- principals visiting us for discussion.
```

calendar는 오늘과 래일의 날자들을 담고 있는 행들을 탐색한다. 이때 이 지령이 받아 들이는 여러가지 형식에 주의를 돌릴 필요가 있다. 현재달은 Mar, March, march로 되여 있다. calendar는 이 모든 형식들을 유효한것으로 인식한다. 하지만 "24th March"의 형식은 calendar가 접수할수 없다.

만일 년을 지정하지 않으면 calendar는 기정적으로 현재의 년을 가정한다. calendar는 또한 주말에 실행될 때에는 좀 다르게 동작하기도 한다. 그때의 《래일》은 주말휴식일만이 아니라 다음주의 첫 로동일도 포함한다.

보는바와 같이 calendar가 받아 들일수 있는 날자형식으로는 여러가지 형식을 택할수 있다. 하지만 날자가 수자들로 지정된다면 mm/dd/yy형식만이 가능하며 mm-dd-yy형식은 리용할수 없다.

주해

calendar는 근본적으로 쉘스크립트이며 쉘프로그람작성에 대하여 상당한 정도로 숙련되면 그 구조를 원만히 해석할수 있을것이다. Linux에서는 이것이 제공되지 않는다.

3.12 수산기프로그람(bc)

UNIX는 두가지 형태의 수산기, 즉 실지의 수산기모양을 가진 xcalc지령과 문자기반의 bc지령을 제공한다. xcalc는 X Window체계에서 유용하며 사용하기 쉽다. bc는 약간 불편하며 극히 강력하지만 UNIX체계에서 무시되는 도구들중의 하나로 되고 있다.

bc를 인수없이 호출할 때에는 유표가 계속 깜박거리면서 아무 일도 생기지 않는다. bc는 인수없이 사용될 때 건반으로부터 입력을 요구하는 지령계(려파기라고 한다.)에 속한다. 아래와 같이 산수식을 건으로 입력하여 보자.

```
$ bc
12 + 5
17                계산후 현시된 값
```

bc는 2개의 수를 더하여 그 출력을 다음행에 보여 준다. bc를 완료하자면 입력의 끝을 표시하는 [Ctrl-d](eof문자)를 사용해야 한다. 사용자는 또한 bc가 여러개의 계산을 함께 진행하도록 요구할수 있다.

```
12*12 ; 2^32          ^는 제곱을 가리킨다
144
4294967296            32bit기계에서 가능한 최대기억기
[Ctrl-d]
$ _
```

이번에는 bc를 다시 시작하여 두 수의 나누기를 진행하여 보자.

```
9/5
1                 소수점아래자리가 생략되였다.
```

bc는 기정적으로 옹근수나누기를 수행하며 사용자는 나누기를 진행하기전에 정확도자리수를 scale에 설정해야 한다.

```
scale=2                소수점아래 두자리로 자른다
17/7
2.42          반올림되지 않았다. 결과는 실지로 2.42857...이다
```

bc는 수체계사이의 변환에 아주 쓸모가 있다. 실례로 망에서 IP주소들을 설정할 때 사용자는 2진수를 10진수로 변환해야 할 필요가 있을수도 있다. ibase를 2로 설정하고 2진수를 입력하여 보자.

```
ibase=2
11001010
202           10진수로 된 출력
```

반대방향의 변환도 가능하다. 이번에는 obase를 가지고 시도하여 보자.

```
obase=2
14
1110          14의 2진수
```

이러한 방법으로 사용자는 수들을 하나의 수체계로부터 다른 수체계에로 변환할수 있다(16을 초과할수 없다). bc는 16진수도 완전무결하게 조작한다.

```
obase=16
14
E             14의 16진수값
```

bc는 또한 프로그람이 완료될 때까지 자기의 값을 보존하는 변수들과 함께 사용될수도 있다. 하지만 bc는 오직 소문자로 된 한 문자변수(a부터 z까지)들만을 지원하기때문에 사용자는 26개의 변수를 사용할수 있다. 변수설정은 아주 간단하며 값평가는 단순히 변수이름을 입력하는것으로 수행된다.

```
x=3 ; y=4 ; z=5
p = x+ y +z
p
12
```

bc는 배렬, 함수, 조건문(if), 순환(for와 while)을 가지는 **모조프로그람작성언어**(pseudo-programming language)이다. 또한 과학기술계산을 위한 서고도 제공한다. 그것은 매우 거대한 수자들을 조작할수 있다. 만일 계산결과가 900자리수내에 있다면 bc는 매 수자를 모두 보여 줄것이다.

이 장에서 설명된 지령들외에도 UNIX에는 파일들을 조작하는 보다 일반적인 목적을 가진 지령들이 많다. 사용자는 그것들이 무엇을 하는지 알려고 할수도 있으며 지어 그 일부를 자기 손으로 직접 시험해보려 할것이다.

- cp, mv, rm를 리용하여 파일의 복사, 이름고치기, 삭제를 진행한다(6.11부터 6.13까지).
- cat, more, 1p를 리용하여 파일을 현시하거나 인쇄한다(6.14, 9.1, 6.16).
- compress와 gzip로 파일을 압축한다(6.19).
- df와 du로 디스크공간의 리용률을 찾아 낸다(6.17, 6.18).

- head와 tail로 파일의 두끝을 추출한다(9.8, 9.9).
- cut를 리용하여 파일을 수직으로 자르고 paste로 2개의 잘라 진 토막들을 붙인다(9.10, 9.11).
- comm, cmp, diff로 두 파일들사이의 차이점을 찾아 낸다(9.5부터 9.7까지).
- sort를 써서 파일내용들을 정돈한다(9.12).

이러한 많은 지령들은 사용법이 유연하며 다음장들에서 취급된다. 이 지령들의 능력을 충분히 평가하려면 이 지령들의 입출력을 조작하는 쉘의 동작특성들을 알아야 한다.

요 약

passwd는 사용자의 통과암호를 변경시키는데 사용되며 그 통과암호는 안전상 리유로 화면에 나타나지 않는다. 통과암호는 무의미한 뜻으로 되여야 하며 일부 체계들은 기억하기 쉬운 통과암호들의 사용을 허용하지 않는다. 체계관리자는 임의의 사용자의 통과암호를 변경시키기 위하여 그의 현재의 통과암호를 알 필요는 없다.

who와 w는 체계상에서 작업하고 있는 사용자들을 보여 준다. 그 지령들은 또한 그들이 가입한 시간과 아무것도 하지 않고 지나보낸 시간들도 보여 준다. w는 CPU리용률의 상세한 정보를 제공하며 매 사용자가 실행하고 있는 지령도 보여 준다.

tty는 사용자에게 그의 말단의 장치이름을 알려 준다. 이것은 항상 /dev등록부안에 있는 파일로 된다.

사용자가 림시 작업장을 떠날 때에는 lock를 사용하여 사용자의 말단을 잠근다. lock가 사용하는 통과암호는 체계가입에 리용되는 통과암호와 같아야 할 필요는 없다.

stty는 여러가지 말단속성들을 설정하는데 사용된다. 사용자는 프로그람을 중단하거나(intr) 후진건으로 본문을 지우며(erase) 파일끝을 표식하는(eof) 건들을 정의할수 있다. 열쇠단어 echo를 설정하여 화면상에 건입력이 현시되지 않게 한다. 아무 동작도 하지 않으면 stty sane을 사용하여 말단을 일정한 표준값들로 설정할수 있다.

script는 사용자의 모든 동작을 별도의 파일에 기록하는 UNIX체계의 기록기이다. 이 지령은 사용자입력과 지령출력을 다같이 한 파일에 저장한다.

clear와 tput clear는 화면을 지우는데 사용된다. tput를 echo와 결합하여 사용하면 본문을 강조하거나 화면의 일정한 위치에 본문을 현시할수 있다.

uname -n은 망지령들이 사용할 기계의 이름을 현시한다.

date는 년, 월, 일 또는 그것들의 결합을 현시할수 있다. 체계관리자는 이 지령을 사용하여 체계날자를 변경시킨다.

cal은 월이나 년의 력서를 만든다.

calendar는 사용자에게 그의 용무를 상기시켜 주는데 사용된다. 금요일에는 《래일》을 주말휴식일과 다음주 월요일을 의미하는것으로 간주한다.

bc는 수산기이다. 이것은 2진수, 10진수, 16진수들을 조작할수 있으며 수체계들사이의 변환도 수행할수 있다. 또한 중간계산결과들을 변수에 보관할수 있고 임의의 정확도를 사용할수 있다.

시험문제

1. passwd지령이 낡은 통과암호를 문의하지 않았다. 언제 이렇게 된다고 생각하는가?
2. 통과암호는 어디에 보관되는가?
3. 1752년 력서를 보시오. 류별난것이 없는가?
4. 아무 사람이나 date지령으로 체계날자를 변경시킬수 있는가?
5. calendar는 어디에서 입력을 얻는가?
6. 어느 지령으로 화면을 지우는가?
7. uname지령을 인수없이 입력하시오. 출력이 무엇을 표현한다고 생각하는가?
8. 체계가 가동해 온 시간을 알아 내자면 어느 지령을 사용해야 하는가?
9. 파일 foo에 가입대화를 어떻게 기록하겠는가?
10. 사용자의 말단장치이름을 어떻게 알아 내는가?
11. 다른 사람이 자기의 말단을 사용하는것을 어떻게 방지할수 있는가?
12. bc가 모든 나누기결과를 소수점아래 3자리수로 현시하게 하려면 어떻게 해야 하는가?
13. 수백명의 사용자들을 가진 체계에서 사용자의 동작을 어떻게 목록으로 얻을것인가?
14. 체계에 가입한 사용자들의 상세한 정보를 현시하지 않고 그들의 이름목록만을 얻어 내자면 어떻게 해야 하는가?

련습문제

1. 현재의 날자를 dd/mm/yyyy형식으로 현시하시오.
2. calendar가 《래일》을 언제 다르게 취급하는가?
3. calendar가 "17th October"를 유효한 날자로 취급하는가?
4. [CapsLock]건이 제대로 설정되였는데도 불구하고 갑자기 건반이 대문자를 현시하고 있다. 무엇을 시도해 볼수 있는가?
5. who출력에서 동일한 가입이름을 한번이상 가질수 있는가?
6. 아무 일도 하지 않고 있는 사용자들을 어떻게 찾아 내는가?
7. 어느 지령으로 기계이름과 조작체계의 판본을 알아 볼수 있는가?
8. 일반사용자가 어느 지령을 사용하여 체계날자와 시간을 변경시키는가?
9. 화면을 어떻게 지우며 유표를 12행 25렬에 배치하려면 어떻게 하여야 하는가?
10. 사용자의 건반조작이 현시되지 않는다면 어떤 수단을 시도하여 볼것인가?
11. 1101001의 10진값을 어떻게 알아 내는가?

제 4 장. vi 및 vim편집기

UNIX체계로 무슨 작업을 하든지 관계없이 사용자는 결국 일정한 C나 Java프로그람들, 또는 쉘(또는 perl)스크립트들을 작성하게 된다. 또한 일부 체계파일들을 편집해야 할 때도 있다. 이러한 작업을 하자면 편집기를 사용할줄 알아야 한다. UNIX는 매우 다방면적인 2개의 편집기들, 즉 vi와 emacs를 제공한다. 이 장에서 우리는 먼저 vi에 대하여 취급하며 다음에 emacs에 대해서도 보게 된다.

vi는 현재 모든 UNIX체계들에서 리용할수 있는 전화면방식의 편집기로서 그 어떤 환경에서도 쓸수 있는 가장 강력한 편집기들중의 하나로 널리 인정되고 있다. 이 편집기는 대학졸업생인 빌 죠이가 만들었으며 그는 후에 싼 마이크로씨스템즈의 창시자의 한 사람으로 되였다. vi는 BSD UNIX에서 처음 출현하였으며 지금은 모든 UNIX체계들에서 표준으로 되고 있다.

vi는 편집작업을 위한 내부지령들을 제공한다. 이 편집기는 거의 완성적인 건반사용을 보장하며 실천적으로 모든 건들이 기능을 가지고 있다. vi는 헤아릴수 없이 많은 기능들을 가지고 있으나 실지로는 사용자가 그 모든 기능에 대하여 알 필요는 없다. 초기에는 작업에 대한 지식이면 된다.

Linux에서 가장 잘 알려 져 있는 vi편집기는 vim(vi improved)이다. 브람 무레나르(Bram Moolenaar)는 이 프로그람에 현저한 3가지의 개선을 가져 왔는데 그것들은 다중창문, 본문강조, 지령리력이다. 한개의 파일이 하나이상의 이름을 가지게 하는 련결기능의 장점으로 하여 Linux에서의 vi를 vim이라고 부른다. 우리는 vim의 기능들에 대하여서도 론의한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 3가지 기능방식들에 대하여 배운다(4.1.1).
- 본문을 입력하고 치환하며 문자들을 조종한다(4.3).
- 작업내용을 보관하고 파괴로부터 회복하며 vi에서 탈퇴한다(4.2, 4.4).
- 반복인자의 의미를 리해한다(4.6).
- 파일과 행을 따라 항행한다(4.8).
- 연산자-지령결합을 사용하여 본문을 삭제, 이동 및 복사한다(4.9, 4.10).
- 대소문자를 포함한 본문을 변경한다(4.11).
- 마지막지령을 반복 또는 취소한다(4.12, 4.13).
- 파일에서 문자렬을 탐색하고 행에서 문자를 탐색한다(4.14).
- 정규식들을 포함하도록 탐색기구를 확장한다(4.15).
- 패턴을 다른것으로 치환한다(4.16).
- 여러개의 파일들을 편집하며 한 파일로부터 다른 파일로 절환한다(4.17).
- 본문에 표식을 주고 임의의 표식에 접근한다(4.18).
- UNIX지령을 실행(려파)시켜 화면본문을 변경시킨다(4.19).
- 여러개의 본문부분들을 개별적인 완충기들에 복사한다(4.20).
- 9개까지의 완전행삭제를 회복한다(4.21).
- 생략과 건넘기기를 정의하고 변수들을 설정하여 편집기의 환경을 전용화한다(4.22, 4.23, 4.24).

4.1 vi의 기초

vi작업은 지령 vi를 파일이름과 함께(또는 파일이름없이) 호출하는것으로부터 시작된다.

```
vi index.html
```

사용자에게는 완전히 빈 화면이 주어 지며(그림 4-1) 매행은 물결표(~)로 시작된다. 이것은 vi가 빈 행을 가리키는 방법이다. vi는 보통 말단에서 리용할수 있는 25개 행가운데서 24개를 본문편집에 사용한다. 마지막행은 사용자가 본문에 작용하는 지령들을 입력하기 위하여 예약되여 있다. 이 행은 또한 체계가 통보문을 현시하는데도 사용된다. 파일이름은 통보문 "index.html" [New file]로 이 행에 나타나게 된다.

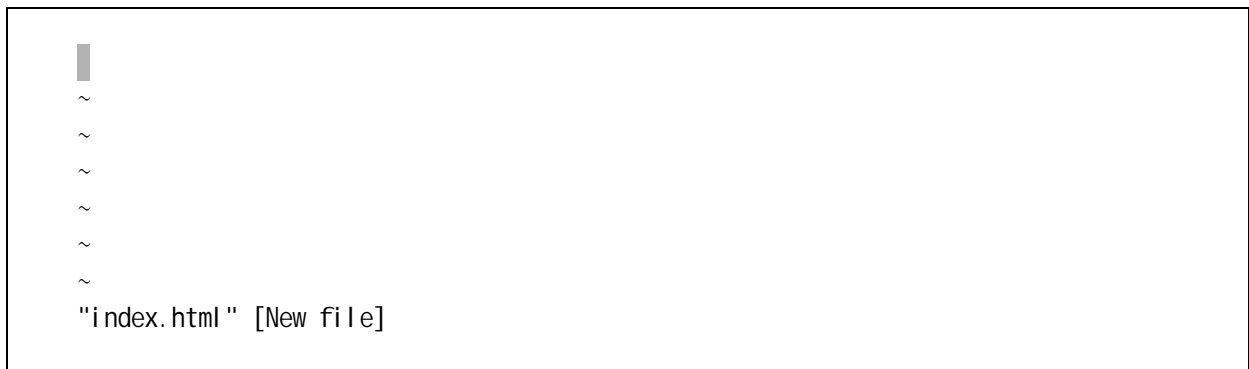



그림 4-1. vi화면

4.1.1 3가지 기능방식

vi로 파일을 열면 유표가 화면의 왼쪽웃구석에 배치된다. 이때 사용자는 **지령방식**(Command Mode)에 있다고 말할수 있다. 이 방식에서 사용자는 지령들을 보내여 본문상에 작용하게 한다. 건반을 누르는것이 화면에 나타나지는 않지만 유표를 다음행으로 이동시키거나 행을 삭제하는것과 같은 기능이 수행될수 있다. 사용자는 본문을 입력하거나 치환하기 위해서는 이 지령방식을 사용할수 없다.

이 단계에서 사용자가 알아 두어야 할 지령방식의 두가지 기능이 있는데 그것은 [Spacebar]와 [Backspace]건들의 역할이다. [Spacebar]는 유표를 한 문자만큼 앞으로 가져 가며 [Backspace](또는 [Ctrl-h])는 한 문자만큼 뒤로 보낸다. 이 방식에서는 후진건이 본문을 전혀 지우지 못한다.

본문을 입력하자면 지령방식으로부터 **입력방식**(Input Mode)으로 들어 가야 한다. 입력방식으로 가게 해주는 건이 10개 있으며 이 방식에서는 사용자가 입력하는것들이 화면에 현시된다. 뿐만아니라 이 방식에서는 후진건이 유표가 통과해 가는 모든 문자들을 지워 버린다. 이 방식에서 벗어 나려면 [ESC]건을 눌러야 한다.

사용자는 자기가 진행한 작업을 보관하고 vi에서 탈퇴하거나 편집중의 다른 파일로 절환해야 한다. 때로는 파일에서 전역적인 치환을 해야 할 필요가 있게 된다. 그런데 이 두 방식들은 어느 하나도 이러한 작업을 전혀 하지 못한다. 사용자는 **최종행방식**(Last Line Mode 또는 ex Mode)을 사용해야 하며 거기서 화면의 마지막행에 지령을 입력해야 한다. 지령방식의 일부 기능들은 최종행방식에서도 사용될수 있다.

여기로부터 우리는 vi가 작업하는 3가지 방식들에 대하여 개괄해 볼수 있다.

- 지령방식 - 건들이 본문에 작용하는 지령들처럼 사용되는 방식.

- 입력방식 - 눌리운 건들이 본문으로서 입력되는 방식.
- 최종행방식 또는 ex방식 - 본문에 작용할 지령들이 화면의 마지막행에 입력되는 방식.

이 3가지 방식들과 쉘사이의 관계를 그림 4-2에 보여 준다.

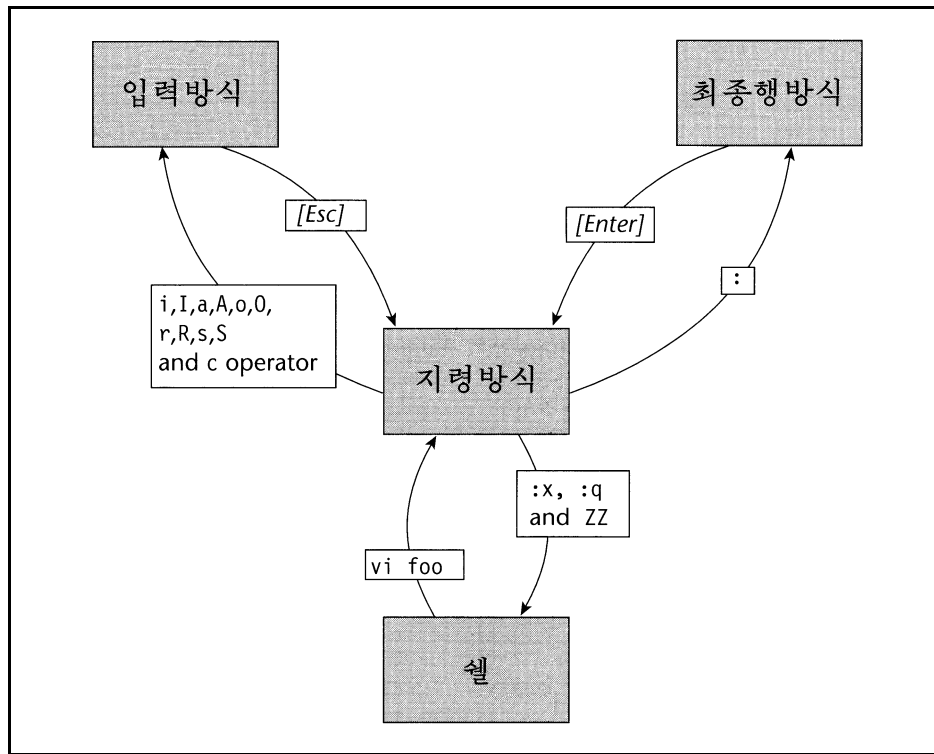



그림 4-2. vi의 3가지 방식들

4.1.2 .exrc파일

vi의 동작은 구성파일에 의하여 조종되며 vi는 그 파일을 시동시에 읽어 들인다. 많은 UNIX지령들이 자기들의 구성파일을 가지고 있으며 vi가 사용하는 구성파일은 .exrc이다. 사용자는 가입시에 배치되는 홈등록부에서 이 파일을 발견할수도 있다. 마지막행방식에서 사용되는 많은 지령들도 이 파일에 배치될수 있다.

ls지령은 특정한 선택항목(-a)과 함께 사용되지 않는한 .exrc파일을 현시하지 않는다. 파일이 여전히 보이지 않으면 사용자는 자기의 파일을 만들거나 체계관리자에게 기정파일을 요구할수 있다. 우리는 앞으로 이 파일을 사용하여 vi를 전용화할것이다.

주의

작업을 계속하기에 앞서 건반에 있는 [CapsLock]건이 켜 있지 않는가를 확인해 볼 필요가 있다. vi지령들은 대소문자를 구분한다. 즉 지령 a는 A와 다르다. 본문블로크를 대문자로 입력하기 위해 [CapsLock]를 사용하는 경우에도 본문입력을 끝낸 다음에는 그 건을 꺼놓았는지 확인하여야 한다

4.2 최종행방식에서 vi의 탈퇴

vi나 다른 임의의 편집기를 사용하여 파일을 편집할 때 본래의 파일은 그러한 편집으로 하여 지장을 받지 않는다. 편집기는 **완충기**(buffer)에 배치된 파일의 복사본을 가지고 작업한다. 이것은 순수 디스크

상에서 그 파일과 련관되여 있는 림시저장령역이다. 흔히 사용자는 완충기내용을 디스크파일에 써넣는것으로써 자기의 작업내용을 보관해야 한다. 영구적인것은 완충기가 아니라 디스크파일이다.

사용자는 또한 어떻게 편집기로부터 벗어 나는가에 대해서도 알고 있어야 한다. 많은 량의 본문을 입력한후에 어떻게 파일을 보관하고 탈퇴하는가 또는 어떻게 모든 변경을 포기하고 탈퇴하는가를 말해줄 사람이 주위에 없어서 그 기계를 재기동해야 한다면 그것은 정말로 비극이다.

우리가 파일을 보관한다고 해도 이것은 완충기를 보관한다는 의미로 인식된다. 보관과 탈퇴(quit)는 최종행방식에 의하여 조종된다. 이 방식에서 모든 지령들앞에는 두점(:)이 놓이며 뒤에는 [Enter]건이 따르게 된다.

4.2.1 보관 및 탈퇴(:wq와 :x)

최종행방식은 보관 및 탈퇴를 위한 두가지 방법 즉 :x와 :wq를 제공한다. 첫번째 방법이 보다 적은 건조작을 요구하므로 우리는 이 방법을 사용하도록 하자. 지령은 작업을 보관한 다음(즉 변경된 완충기를 디스크에 써넣은 다음) 사용자를 쉘에로 돌아 가게 한다.

```
:x[Enter]                    먼저 지령방식에 있어야 한다
"index.html", 8 lines, 303 characters
$ _
```

먼저 :(최종행방식프롬프트)을 입력하고 뒤따라 x와 [Enter]건을 누른다. 이것은 최종행방식의 지령으로서 사용자가 :을 입력하면 그것이 화면의 마지막행에 나타나는것을 볼수 있다. [Enter]를 누른 다음 파일이 보관되고 vi에서 탈퇴할 때에만 통보문이 출현한다.

참고

보관 및 편집기탈퇴의 가장 좋은 방법은 :x나 :wq보다 지령방식의 지령인 ZZ를 사용하는것이다. 이 방식에 들어 가기 위하여 먼저 [Esc]를 사용하는것이 필요하다.

4.2.2 편집의 무시(:q)

편집을 무시(abort)하거나 완충기를 보관하지 않고 편집방식에서 탈퇴하는것도 가능하다. q(quit)지령은 완충기가 변경되지 않았을 때에만 편집기에서 탈퇴하게 한다.

```
:q[Enter]                    이번에는 통보문이 없다!
$ _
```

vi는 또한 안전체계를 가지고 있으며 변경된 파일을 보관하지 않고 예상치 않게 무시하는것을 방지한다. 이제 동일한 지령을 변경된 파일을 보관하지 않은 상태에서 다시 시도하여 보자.

```
:q[Enter]
No write since last change (use ! to override)
```

통보문은 사용자가 변경된 내용을 보관하지 않고 탈퇴를 시도하고 있다는것을 알려 주고 있다. 그래도 변경내용을 포기(abandon)하려고 한다면 다음의 지령을 사용할수 있다.

```
:q![Enter]                   !는 많은 안전장치설정들을 재정의한다
```

이 지령은 아무런 질문도 제기함이 없이 그리고 완충기상태에는 무관계하게 사용자를 프롬프트에로 되돌려 보낸다. 중요한 보관 및 탈퇴지령들을 표 4-1에 보여 준다.

표 4-1. vi에서 보관 및 탈퇴지령들

지 령	기 능
:w	파일을 보관하고 편집방식을 유지한다
:x	파일을 보관하고 편집방식에서 탈퇴한다
:wq	우와 같다
:q	파일이 변경되지 않았을 때 편집방식에서 탈퇴한다
:q!	변경을 취소하고 편집방식에서 탈퇴한다
:w n2w.pl	파일 n2w.pl로 보관한다(MS Windows에서의 Save As...와 같다)
:w! n2w.pl	우와 같으나 존재하는 파일에 덧쓴다
:w >> note1	현재파일의 내용을 파일 note1에 추가한다
:n1,n2w build.sql	n1부터 n2까지의 행들을 파일 build.sql에 써넣는다
:.w build.sql	현재행을 파일 build.sql에 써넣는다
:$w build.sql	마지막행을 파일 build.sql에 써넣는다
:sh	UNIX쉘에로 림시탈퇴한다(vi에로 돌아 가려면 exit를 사용한다)
[Ctrl-z]	현재작업을 림시중지하고 UNIX쉘에로 탈퇴한다(쉘이 일감조종을 지원하는 경우에만 유효하며 vi에로 돌아 가려면 fg를 사용한다)

주해

일반적으로 최종행방식지령들이 !와 함께 사용되면 편집무시형식을 의미한다. 그것은 현재의 파일을 보관하지 않고 다른 파일로 절환하거나 그 파일의 마지막으로 보관된 판본을 다시 읽어 들이는데 사용될수 있다. 별개의 파일에 덧쓰기하는데도 사용할수 있다.

4.3 본문의 삽입과 치환

본문을 입력할수 있도록 하자면 편집기의 방식을 초기의 지령방식으로부터 입력방식으로 바꾸어야 한다. 이 방식으로 들어 가는데는 몇가지 방법이 있다. 이 방법들은 사용자가 건으로 입력하려는 입력형태에 따른다. 하지만 그 어떤 경우에도 [Esc]건을 누르면 그 방식이 완료된다.

참고

vi를 처음 리용하여 보는 사용자들은 vi를 불러 낸후에 다음의 최종행방식지령을 주고 편집을 시작하는것이 좋다.

```
:set showmode[Enter]          먼저 [Esc]를 누른다
```

:x를 입력할 때와 같은 방법으로 입력한다. 이 지령은 vi환경을 위한 한개의 파라메터를 설정하고 입력방식이 호출될 때마다 적절한 통보문을 현시한다. 통보문은 화면의 제일 아래행에 출현한다. UNIX의 일부 판본들은 이 방식으로 구성된 vi를 가지고 있으며 그러한 경우에는 이 설정이 필요 없다

4.3.1 본문의 삽입(i, I)

가장 단순한 입력형태는 본문의 삽입이다. 파일에 본문이 있건없건 관계없이 vi가 호출되면 유표는 언제나 첫번째 행의 첫 문자에 위치한다. 이 자리에 본문을 삽입하려면 i를 누른다.

```
i               본문이 있으면 오른쪽으로 밀려 갈것이다
```

화면에 그 문자가 나타나지는 않지만 이 건을 누르게 되면 지령방식으로부터 입력방식으로 전환한다. 시작할 때 표시방식이 설정되였으므로(:set showmode로) 마지막행에서 INSERT나 INSERT MODE라는 단어들을 보게 될것이다. 이제는 사용자가 누르는 건들이 화면에 본문으로 입력된다. 그림 4-3에서와

같이 몇개 행의 본문을 삽입하여 보자(매행의 뒤에 [Enter]를 누르면서).

```
This is the vi editor[Enter]
Itis slow in getting started but is quite powerful[Enter]
It operates in three modes[Enter]
All the features of ex are also available[Enter]
You can even escape to the UNIX shell[Enter]
It maintains 26 buffers for storing chunks of text
~
~
~
~
~
```

그림 4-3. vi에 본문을 입력하기

이제는 유표가 마지막행의 마지막문자에 위치하고 있다. 이것을 **현재행**(current line)이라고 한다. 유표가 위치하고 있는 문자를 **현재유표위치**(current cursor position)라고 한다. 만일 이 행에서 잘못된 것을 발견하게 되면 [Backspace]를 사용하여 삽입한 본문을 한번에 한 문자씩 지워 버릴수 있다. [Ctrl-w]를 사용하여 이전의 단어도 지워 버릴수 있다.

몇개의 행들을 입력한후에는 [Esc]를 눌러 지령방식(처음 있던 곳)으로 돌아 가야 한다. 작업내용이 아직 보관되지 않았다는데 주의를 돌려야 한다. 바로 여기서 사용자는 방금전에 설명한 보관 및 탈퇴지령들을 사용할수 있으며 바란다면 편집기에서 탈퇴할수 있다.

우리는 i로 삽입을 시작하였고 유표위치의 왼쪽에 본문을 넣었다. 만일 본문이 존재하는 곳에 위치한 유표를 가지고 i지령을 호출한다면 그 오른쪽에 있는 본문은 덧쓰기되지 않고 앞으로 밀려 나가게 될것이다.

i를 사용하면 행의 임의의 곳에 본문을 삽입할수 있지만 I는 오직 행의 시작부분에만 본문을 삽입한다. 유표가 행의 끝에 놓여 있다고 해도 I를 가지고 행시작점에 이동한 다음 본문을 삽입할수 있다.

I	유표는 먼저 시작행으로 이동해 간다

이것은 또한 현재행의 시작점에로 이행하는 빠른 방법이라고 말할수 있다. 그림 4-4에 i와 I에 의한 본문의 삽입을 보여 준다. 이 장과 다음장의 모든 그림들에서 사용하고 있는 회색칸은 유표를 표시하며 □은 공간(공간이 명백치 못할 때 사용)을 의미한다.

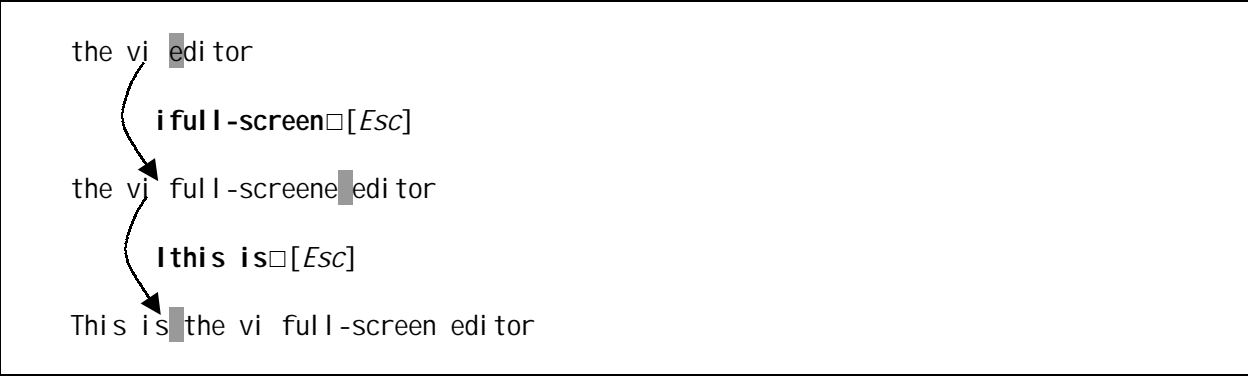



그림 4-4. i와 I지령들로 본문삽입

4.3.2 본문의 추가(a, A)

본문입력에는 다른 방법들도 있다. 본문을 유표가 위치한데로부터 오른쪽에 첨가하자면 a를 사용하고 그뒤에 원하는 본문을 입력한다.

a 현존본문은 오른쪽으로 밀려 난다

여기서는 i에서와 달리 본문을 유표오른쪽에 배치한다. 편집이 끝나게 되면 [Esc]를 누른다.

i와 a를 리용하여 이러한 방법으로 몇개 행의 본문을 추가할수 있다. 하지만 a도 역시 I와 꼭 반대되는 방법으로 동작하는 대문자지령 A를 가지고 있다. 행의 끝에 본문을 추가하려면 A를 사용한다.

A 단락을 계속하는데 적합하다

I와 같이 이것도 행의 끝으로 이행하여 거기에 본문을 추가하는 가장 빠른 방법을 의미한다. 사용자는 자기가 중단했던 곳으로부터 본문을 계속하기 위하여(례하면 단락에 문장을 추가하기 위하여) 이것을 사용한다. a와 A지령들의 사용법을 그림 4-5에 보여 준다.

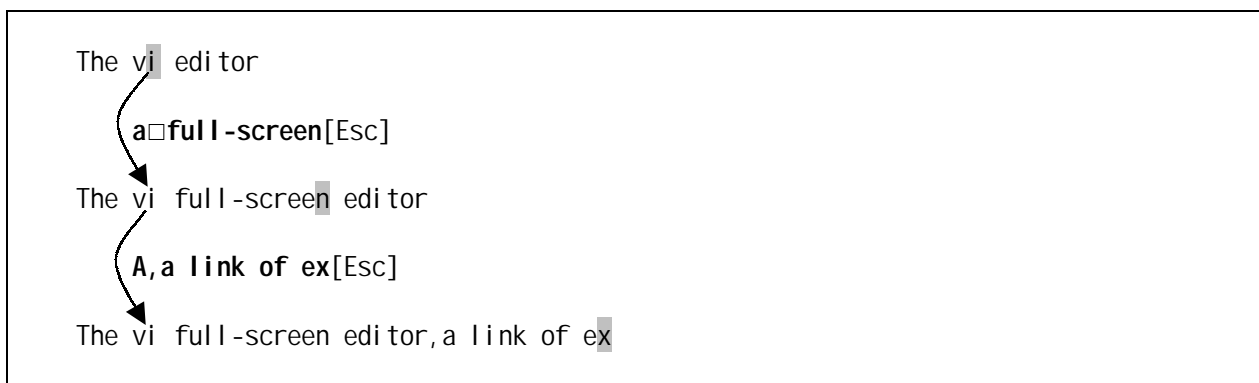



그림 4-5. a와 A지령으로 본문에 삽입

4.3.3 새로운 행을 시작하기(o, O)

행의 임의의 장소에 유표를 놓고 o를 누르는것으로 새로운 행을 펼칠수 있다(행삽입).

o 아래에 새로운 행을 펼친다

이것은 현재행의 아래에 빈 행을 삽입한다. 현재행우에 행을 삽입하려면 O를 사용한다.

O 우에 새로운 행을 펼친다

어느 경우에나 showmode설정은 사용자가 입력방식에 있다는것을 알려 준다. 사용자는 여러개의 행들을 펼쳐서 요구되는 량의 본문을 자유롭게 입력할수 있다. 본문입력를 완성한 다음에는 [Esc]를 누른다. 그림 4-6에 o와 O의 사용법에 대하여 보여 준다.

4.3.4 본문의 치환(r, R, s, S)

r, R, s, S를 리용하여 본문을 치환한다. 어떤 한개의 문자를 다른 문자로 치환하자면 r를 사용하고 뒤이어 교체해 넣을 문자를 입력한다. 이 방법으로는 한 문자만을 치환할수 있다.

r 여기서는 [Esc]가 요구되지 않는다

r가 눌리울 때 vi는 순간적으로 지령방식으로부터 입력방식으로 절환한다. 이것은 치환문자가 입력되는 즉시 지령방식으로 돌아 온다. vi는 한 문자만을 요구하므로 r(와 그 문자)를 사용한 다음에 [Esc]를 누를 필요가 없다.

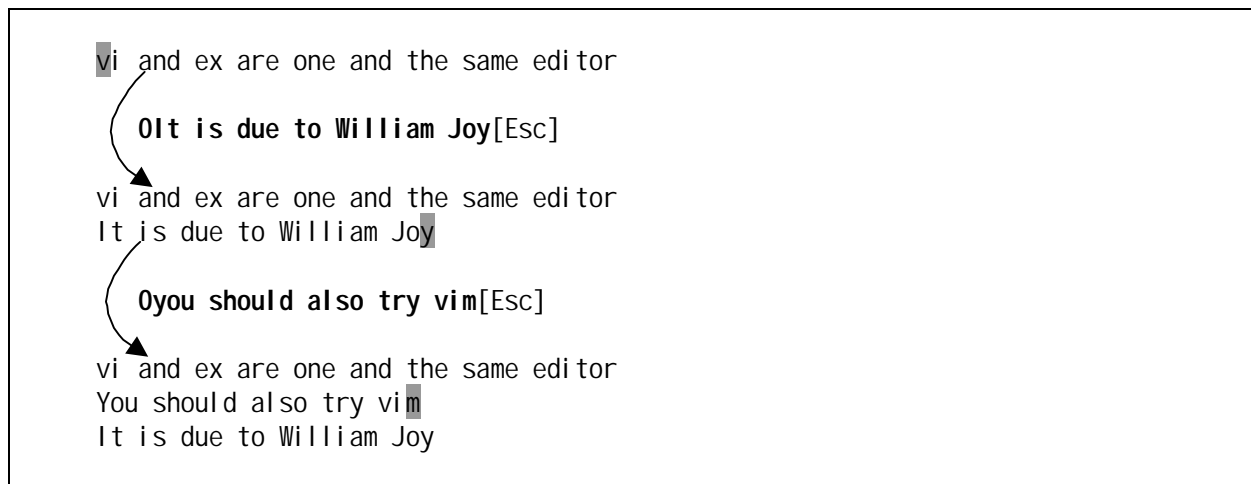



그림 4-6. o와 O지령들로 새로운 행의 시작

한개이상의 문자를 치환하자면 R를 사용하고 그뒤에 그 본문을 입력한다.

R 유표가 오른쪽으로 이동하면서 본문이 치환된다

이것은 emacs의 덧쓰기방식으로서 유표가 앞으로 전진하는데 따라 현존본문이 덧쓰기된다. 하지만 이 치환은 오직 현재행에만 제한된다. 그림 4-7에 지령 r와 R의 치환을 보여 준다.

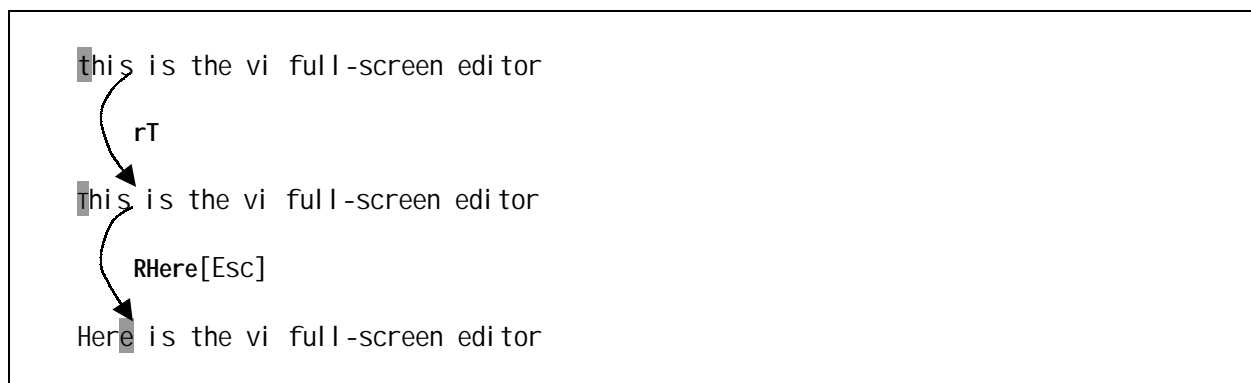



그림 4-7. r와 R지령들로 본문에 치환

많은 경우 사용자는 한개의 문자를 여러 문자본문으로 치환해야 할 필요가 있게 될것이다. 그때에는 유표를 해당 위치에 가져 간 다음 s를 누른다.

s 한개를 여러개로 치환한다

이제는 요구하는 정도의 본문을 입력하고 [Esc]를 누른다. s지령은 문자 a를 the로 변경시키는데 리용할수 있다. 오른쪽에 있는 본문은 앞으로 밀려 나간다.

S는 유표위치에 상관없이 행전체를 치환한다. 행내용을 변경시키기 위하여 유표를 행시작점에 가져갈 필요는 없다. 현재의 위치에 유표를 그대로 두고 S를 누른다.

S 현재행이 없어 진다

행전체가 시야에서 사라지게 된다. 본문을 입력한 다음 [Esc]를 누른다. 그림 4-8에 s와 S의 본문치환에 대하여 보여 준다.

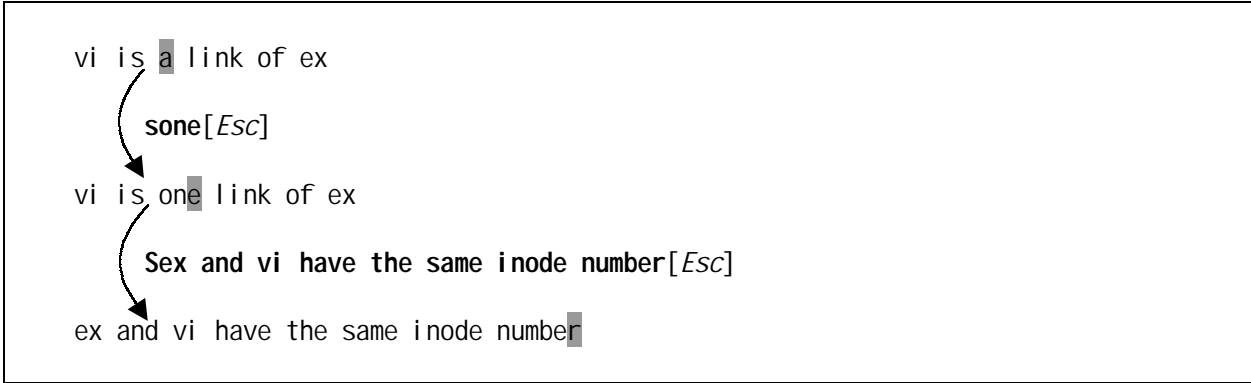



그림 4-8. s와 S지령에 의한 본문치환

이제는 우리가 10가지 방법으로 입력방식에 들어 갈수 있게 되였다. 사용자는 이 10개의 지령들로 수행할수 없는 많은 작업을 진행하기 위해 지령방식으로 절환해야 한다. 이 10개의 건들이 가지고 있는 기능들을 표 4-2에 보여 준다. 보다 정확히 말한다면 입력방식으로 들어 가는데는 c연산자를 사용하는 한가지 방법이 더 있다. 이것을 조작하려면 먼저 연산자들에 대하여 리해하여야 할것이다.

주의

입력방식으로부터 지령방식으로 절환하려면 [Esc]건을 누른다. 만일 이것을 놓치면 지령방식의 모든 지령들이 본문입력으로서 나타나게 될것이다. [Esc]건을 반복해서 눌러도 vi에 그 어떤 변화를 가져 오지는 않지만 필요없이 건을 눌렀을 때에는 경고음으로 알려 주는 기능이 내장되여 있다.

표 4-2. vi에서의 입력방식지령들

지 령	기 능
i	유표의 왼쪽에 본문을 삽입한다
I	행의 시작점에 본문을 삽입한다
a	유표의 오른쪽에 본문을 추가한다
A	행의 끝에 본문을 추가한다
o	아래에 행을 삽입한다
O	우에 행을 삽입한다
rch	유표아래의 한개 문자를 ch로 교체한다([Esc]가 필요 없다)
R	유표로부터 오른쪽으로 본문을 교체한다(존재하는 본문에 덧씌여 진다)
s	유표아래의 한개 문자를 여러개의 문자로 교체한다
S	전체 행을 교체한다

4.3.5 조종문자의 입력([Ctrl-v])

만일 사용자가 자기의 인쇄기나 말단에 몇개의 확장문자렬(escape sequence)들을 전송하기 위한 쉘스크립트들을 작성한다면 조종문자들을 입력할 필요가 있을것이다. vi에서 이 문자들의 일부는 직접 입력될수 있지만 일반적으로는 그 조종문자앞에 또 하나의 조종문자가 선행되여야 제대로 해석될수 있다.

vi는 임의의 조종문자앞에 [Ctrl-v]를 놓는다. 실례로 [Ctrl-h]를 입력하려면 먼저 [Ctrl-v]를 누르고 그다음에 [Ctrl-h]를 눌러야 한다. 그때 화면상에서 다음과 같은것을 보게 될것이다.

```
^H                여기에는 오직 한개의 문자가 있다
```

마치 ^(탈자기호)와 H가 따로따로 보이는것 같지만 여기에는 오직 한개의 문자만이 있을뿐이다. 사용자는 H가 아니라 ^에 유표를 배치할수 있으며 이것이 바로 조종문자들을 식별하는 방법이다.

[Esc]문자를 입력할 때 동일한 기술이 적용될수 있다. [Ctrl-v][Esc]건을 누르면 사용자는 다음과 같이 보이는 [Esc]문자를 보게 될것이다.

^[한개의 문자만이 있다

이것도 역시 한개의 문자로서 사용자는 유표를 ^에만 놓을수 있다. 조종문자의 삽입을 그림 4-9에 보여 준다.

그림 4-9. 조종문자의 삽입

주해

만일 현재 리용하고 있는 vi판본에서 우에서 지적한대로 [Esc]문자가 입력되지 않으면 다음과 같이 시도하여 볼수 있다. 자기가 가지고 있는 vi에서 [Esc]문자를 입력할수 있다면 [ctrl-v][ctrl-[]을 리용한다. [Esc]문자는 입력방식을 완료시키는 지령이기때문에 vi에서 특수하게 취급될 필요가 있을수 있다.

4.4 본문의 보관(:w)

우리는 편집내용을 보관하고 탈퇴하는 방법과 보관하지 않고 탈퇴하는 방법에 대하여서는 이미 배웠다. vi는 편집기를 탈퇴함이 없이 사용자의 작업내용을 보관하기 위한 지령들을 제공한다. 이 방식에서 중요한 보관 및 탈퇴지령들을 표 4-1에 보여 준다.

파일을 보관하고 편집방식을 유지하려면 w(write)지령을 사용한다.

```
:w
"sometext", 8 lines, 275 characters
```

통보문은 파일이름과 함께 보관된 행과 문자들의 개수를 보여 준다. w지령과 함께 다른 파일이름을 지정할수도 있다.

```
:w anotherfile
"anotherfile" [New File] 8 lines, 275 characters written
```

이 경우에 그 내용은 별도로 anotherfile에 씌여 진다. 그렇지만 vi가 마지막행에 anotherfile을 보여 주고 있다고 해도 사용자의 현재파일은 여전히 sometext로 된다. Windows사용자들은 이 선택적인 파일보관기능이 File차림표의 Save As...선택항목과 다르다는데 대하여 주의하여야 한다(Save As...선택항목도 다른 이름으로 파일을 보관하지만 새로운 파일을 현재파일로 되게 한다).

참고

때때로 단순한 :w는 파일을 보관하지 못할수도 있다. 파일의 쓰기가 허용되여 있지 않거나 다른 사람에게 소유되여 있을수도 있다. vi는 여전히 본문을 입력하게 하겠지만 그것을 보관하기 위해서는 다른 파일이름을 가지고 :w를 사용해야 한다.

4.4.1 선택된 행들의 쓰기

w지령은 한개 또는 두개의 **행주소**(line address)와 함께 사용될수 있다(여기서 행주소란 한개 또는 그이상의 행들을 표현하는 한개 또는 두개의 수이다). 만일 한개의 주소가 사용되면 w는 지정된 행을 파일에 써넣는다. 두개의 주소가 사용되면 w는 그 행번호로 지정되는 묶음을 써넣는다. 사용자는 그것들을 아래와 같이 사용한다.

```
:5w n2words.p1              파일 n2words.p1에 5번째 행을 쓴다
:10,50w n2words.p1          같은 파일에 10행부터 50행까지를 쓴다
```

vi는 파일의 현재행과 마지막행을 표시하는 두개의 기호들을 가지고 있다. .은 현재행을 표현하며 $는 마지막행을 표시한다. 실례로 아래의 지령은 현재행으로부터 마지막까지를 파일에 써넣는다.

```
:.,$w n2words.p1
```

행쓰기에는 다른 방법들도 있는데 이에 대해서는 차후에 본다.

주해

최종행방식에서 현재행의 번호는 .(점)으로 표시하며 마지막행은 $로 표시한다. 따라서 :1,$는 파일전체를 가리킨다.

4.4.2 파괴로부터의 회복

파일보관이 언제나 가능한것은 아니다. 전원고장이나 핵심부오유 같은것들에 의해 일부 작업내용을 보관하지 못하는 경우가 있게 된다. 하지만 vi가 디스크상의 완충기를 가지고 작업하므로 때때로 사용자는 작업의 많은 량을 회복할수 있다. 사용자는 파괴(crash)현상이 일어 났다는 통보를 받을것이며 또한 vi자체가 사용자에게 파국적인 사태에 대해 통보할수도 있다. 이 경우에는 편집기를 완료한 다음 -r(회복)선택항목을 가지고 편집기를 다시 기동시킨다.

```
vi -r foo
```

내용을 보면 -r선택항목을 사용하지 않았을 때보다 더 새로운 판본을 보게 된다는것을 알수 있다. :w로 파일을 보관하면 구조작업은 완성된다.

vi의 일부 판본들(특히 Linux)은 vi호출시에 교체파일(.swp확장자를 가진)을 만들며 작업이 끝난 다음에는 그것을 삭제한다. 파괴가 일어 나면 그 파일은 제거되지 않으며 vi는 기동할 때 그 파일이 존재하는가를 검사한다. 일단 회복조작이 끝나면 파일을 수동적으로 삭제하여야 한다(rm지령을 사용하여).

주의

사용자는 매번 완전한 회복을 확신할수는 없다. -r선택항목을 사용할 때 vi는 흔히 사용자에게 피상한 기호들을 보여 줄수도 있다. 그 경우 파일을 보관하지 말고 그대로 탈퇴하여야 한다(:q!를 써서). 이 경우에는 회복이 불가능하며 보통방법으로 vi를 다시 시작하여야 한다.

4.5 UNIX쉘에로의 탈퇴

때때로 사용자에게는 쉘프롬프트를 가져다 주는 **쉘탈퇴**(shell escape:림시적인 탈퇴를 의미한다.)를 만들어야 할 필요가 제기된다. 사용자는 그 프롬프트에서 일정한 작업을 하고 vi로 되돌아 올수 있다. 다음의 최종행방식지령이 그러한 탈퇴기능을 제공한다.

```
:sh
$ _                 프롬프트가 돌아 온다
```

exit 또는 [Ctrl-d]를 누르면 vi로 되돌아 온다. 이 프롬프트에서 vi를 다시 한번 입력하게 되면 사용자는 두개의 vi를 가지게 될것이며 이것은 큰 혼란을 주게 될것이다. ps지령을 사용하여 vi가 여전히 실행되고 있는지 알아 볼수 있다. 만일 exit로 vi화면이 되돌아 오지 않는다면 이것은 vi가 실행되고 있지 않거나 제거되였다는것을 의미한다. 그러한 경우에 사용자는 체계에서 탈퇴될것이다.

주해

$나 %프롬프트로 자기의 존재를 표현하는 쉘은 SHELL변수의 설정에 의하여 결정된다. 이것은 기본적으로 그 사용자를 위해 체계관리자가 설정한 쉘이다. sh은 사실 Bourne쉘을 표현하지만 :sh는 일반적인 쉘탈퇴지령이다. $SHELL의 값에 따라 Korn쉘이나 C쉘, bash로 될수 있다. echo $SHELL을 리용하여 현재 사용하고 있는 쉘을 찾아 낼수 있다.

사용자는 쉘프롬프트를 가져 오지 않고도 한개의 UNIX지령은 실행시킬수 있다. :!를 지령이름과 함께 사용하여 보자. 아래와 같은 방법을 리용하여 편집기내부로부터 날자를 알아 낼수 있다.

```
:!date[Enter]
Sun Feb 13 10:26:53 EST 2000
Press RETURN or enter command to continue
```

여기서 [Enter]를 누르면 본래 있던 편집화면으로 돌아 오게 된다.

참고

쉘에로 림시 탈퇴(escaping)하는데는 또 하나의 방법이 있다. 만일 그 쉘이 일감조종을 지원한다면 사용자는 [Ctrl-z]를 눌러 쉘프롬프트를 볼수 있다. 이것이 동작한다면 fg지령으로 편집기에 되돌아 갈수 있다. fg는 배경일감을 전경에로 보내는 쉘의 내부지령이다.

4.6 반복인자

vi와 emacs의 가장 유명한 특징의 하나는 지령앞에 수자를 붙이는 기능이다. 이것이 수행되면 대부분의 지령들은 그 지시를 여러번 반복하라는것으로 해석한다. 실례로 3s는 유표아래에 있는 다음번 세개 문자를 치환한다. 지령앞에 붙은 이 수자를 **반복인자**(repeat factor) 또는 **계수앞붙이**(preceding count)라고 부른다. 이 책에서는 대체로 반복인자라는 용어를 사용하였다.

사용자가 한행에 20개의 별표들을 련속 삽입하려고 한다고 가정하자. i를 삽입하고 *를 20번 입력하기보다는 간단한 방법을 사용할수 있다.

```
20i*              오직 지령방식에만 해당된다
```

여기에는 [Enter]나 [Esc]의 입력이 뒤따라야 한다. 사용자는 그 행에서 20개의 련속된 별표(*)를 보게 될것이다(그림 4-10).

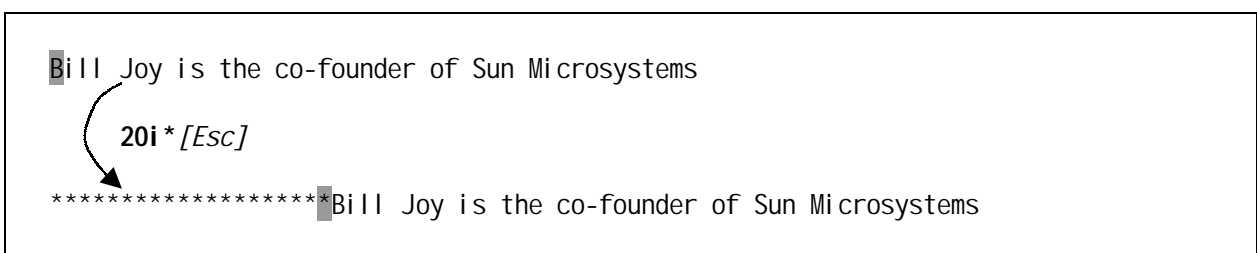



그림 4-10. i지령과 반복인자의 사용

반복인자는 입력방식의 대부분의 지령들뿐아니라 지령방식지령들에도 적용된다. 다음절들에서 우리는 이것이 어떻게 조종과 편집을 더 빠르게 해주는가를 보게 될것이다.

4.7 지령방식

앞으로는 지령방식의 기능들에 대하여 주로 취급하게 될것이다. 이 방식은 사용자가 본문의 입력이나 변경을 끝냈을 때 돌아 오게 되는 방식이다. 이 방식은 항행과 잘라붙이기조작을 수행하는데 의의가 있다. 여기서 건들은 혼자서도 사용되고 결합되여서도 사용된다.

입력방식과는 달리 지령방식에서 건을 누르게 되면 그것이 화면에 나타나지는 않고 단순히 자기의 기능을 수행하기만 한다. 때문에 사용자는 화면에서 변경된 내용들은 볼수 있지만 그 변경을 가져 온 지령은 알수 없다. 초학자들과 emacs사용자들이 이에 대해 짜증을 느낄수도 있지만 실천적으로 해보는 과정에 점차 익숙되게 된다.

주해

표준적인 UNIX에서는 오직 g, K, q, v, V, Z건들만이 기능을 가지지 않는다. 이 건들중 일부는 vim에서 기능을 가진다.

4.8 항행

만일 사용자가 Microsoft Word와 같은 문서편집기를 사용한다면 사용자는 유표이동건들을 널리 사용해야 한다. 만일 사용자가 말단들을 사용하고 있지 않다면 이 건들은 vi나 emacs를 사용할 때 동일한 방법으로 동작한다. 사실상 사용자가 PC나 워크스테이션에서 UNIX를 사용하고 있다면 본문의 이 점에 도달할 때까지 같은 건들을 계속 사용하고 있을수 있다. 그러나 말단들과 같이 UNIX의 더 오랜 일부 판본들은 이 건들을 인식하지 못한다.

vi는 4개의 방향으로 유표를 이동시키는 지령(h, l, j, k)들을 가지고 있다(표 4-3). 이것들은 건반의 중간행에 한줄로 배치되여 있다. 유표의 수평수직이동에 대하여 그림 4-11에 보여 준다.

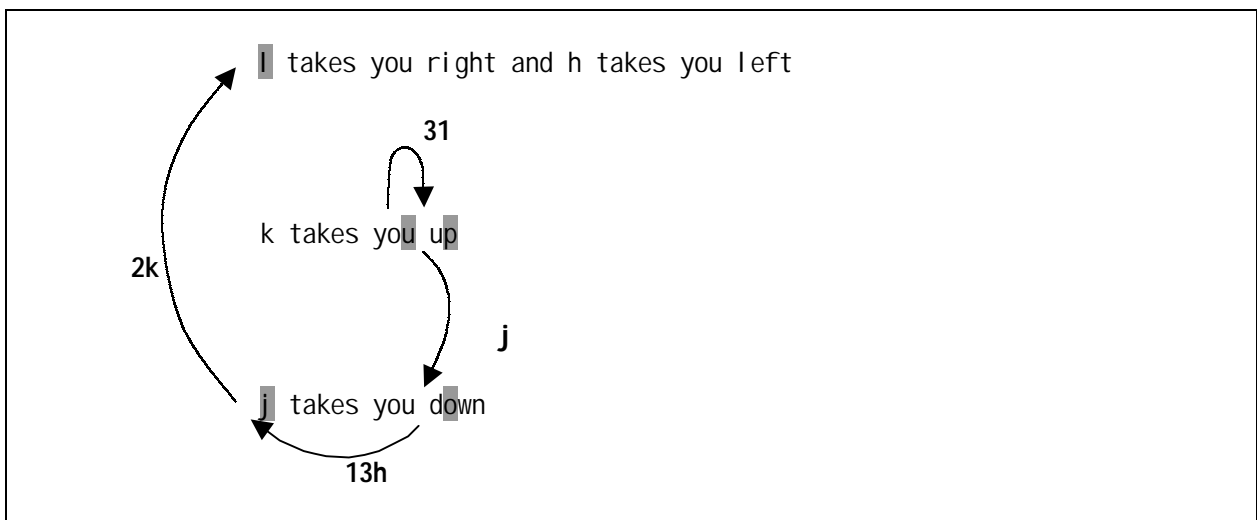



그림 4-11. h, l, j, k로 수평수직이동

표 4-3. vi에서 수평수직이동

지 령	기 능
h 또는 [Backspace]	유표를 왼쪽으로 이동시킨다
l 또는 [Spacebar]	유표를 오른쪽으로 이동시킨다
5l	5문자만큼 오른쪽으로 이동시킨다
k	유표를 우로 이동시킨다
10k	10행우로 이동시킨다
j	유표를 아래로 이동시킨다

여기서와 다음제목에서 취급되는 모든 **항행**(navigation)지령들도 반복인자를 사용한다. 3h는 유표를 왼쪽으로 3자리 이동시키고 7k는 유표를 7줄 우로 이동시킨다.

4.8.1 화면흘리기

사용자는 조종건들을 사용하여 자기 창문에 현시되는 본문의 보기구역(view)을 변경시킬수 있다. vi는 반페지 또는 한 페지단위로 흐르게 할수 있게 한다. 표 4-4에 4개의 중요흘리기(scrolling)지령들과 화면을 다시 그리는 지령을 보여 준다.

표 4-4. vi에서 화면흘리기지령들

지 령	기 능
[Ctrl-f]	전화면 앞으로 흘러 보낸다
5[Ctrl-f]	5개의 전화면만큼 앞으로 흘러 보낸다
[Ctrl-b]	전화면 뒤로 흘러 보낸다
[Ctrl-d]	반화면 앞으로 흘러 보낸다
[Ctrl-u]	반화면 뒤로 흘러 보낸다
[Ctrl-l]	화면을 다시 그린다(반복인자가 없다)

일정한 원인으로 화면이 일부 헝클어 졌을 때 [Ctrl-l]을 사용하여 화면을 다시 그린다. 여기에도 반복인자가 사용될수 있다. 5[Ctrl-f]는 앞으로 5개의 화면을 흘러 보내며 10[Ctrl-u]는 뒤로 10개의 반화면을 흘러 보낸다.

주해

emacs사용자들은 지령방식에 있지 않는 경우에는 이 조종건들을 가지고 앞뒤로 이동할수 없다는데 대해 주의해야 한다.

4.8.2 행의 시작과 끝(0, |, $)

행의 시작점과 끝점으로 이동하는것은 일반적인 요구이다. 이것은 지령 0(령), |, $에 의하여 조종된다. 행의 첫 문자로 이동하려면 0이나 |를 사용한다.

```
0 또는 |                30|는 유표를 30렬에로 이동시킨다
```

행의 마감에로 이동하자면 $를 사용한다.

```
$
```

이 두 지령들과 단어단위를 사용하는 지령(b, e, w)들의 사용법에 대하여 그림 4-12에 보여 준다.

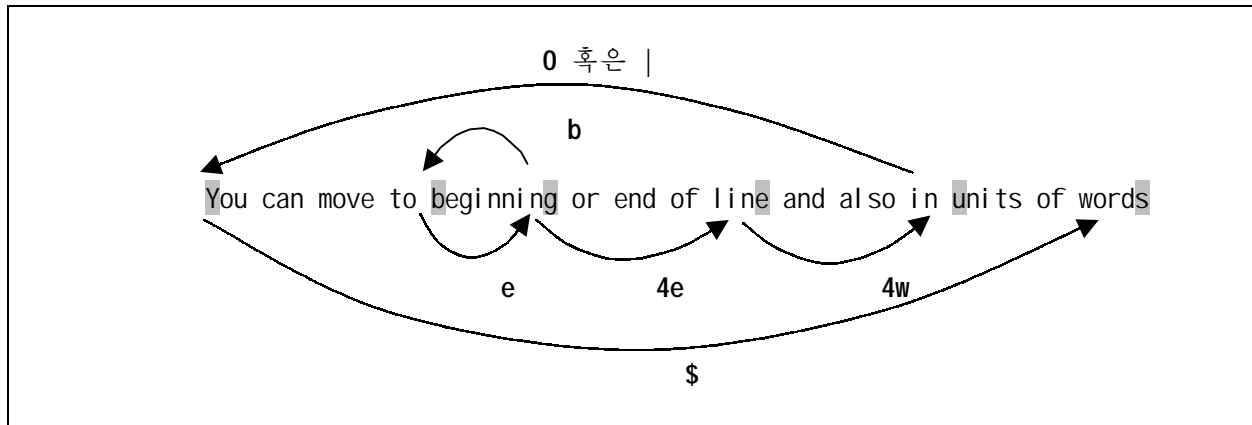



그림 4-12. b, e, w, 0, $에 의한 항행

4.8.3 단어단위의 이동(b,e,w)

한 문자에 의한 이동이 언제나 충분한것은 아니며 사용자는 흔히 더 빨리 이동할 필요가 제기된다. vi는 단어(word)를 항행단위로서 리해하며 단어단위로 유표를 이동하는 지령들을 제공한다. vi에서 단어는 그것이 사용되는 문맥에 따라 서로 다른 의미를 가진다.

단어지향이동(word-oriented movement)은 b, e, w지령들에 의하여 수행된다. 그것들의 기능을 아래에 보여 준다.

b - 단어시작점으로 후진한다.

e - 단어끝으로 전진한다.

w - 단어시작점으로 전진한다.

반복인자는 항행속도를 빠르게 한다. 실례로 5b는 유표를 5개 단어의 뒤로 가게 하며 3w는 유표를 3개 단어 앞으로 가게 한다.

여기서 단어는 단순히 자모와 수자, _(밑줄)문자의 렬이다. bash도 한 단어이며 sh_profile도 역시 마찬가지이다. tcp-ip는 3개의 단어로 되며 -부호도 한개의 단어로 된다.

B, E, W지령들은 점을 뛰여 넘는다는것을 제외하고는 소문자에서와 류사한 기능을 수행한다. 여기서는 단어의 정의도 변화되지만 별로 중요하지 않으므로 상세한 내용은 무시하기로 한다.

4.8.4 행에로의 이동(G)

만일 사용자가 현재 유표가 위치한 행의 번호를 알려고 할 때에는 간단히 [Ctrl-g]를 누른다. 이때 사용자는 다음과 같은 행을 볼수 있다.

```
"salslip.pl" [Modified] line 403 of 541 -74%-
```

사용자는 즉시에 파일이름과 현재의 행번호, 파일의 전체 행개수와 퍼센트로 표시된 그 페지의 상대위치를 알게 된다. vi의 일부 판본(vim와 같은)들은 화면의 오른쪽밑부분에 현재행번호와 렬번호를 항상 보여 준다. 그때에는 [Ctrl-g]가 필요 없다.

사용자는 G지령을 사용하여 유표를 특정한 행에로 이동시킬수 있다. 이 지령앞에는 행번호가 놓인다. 즉 40행으로 이동하려면 40G를 사용한다.

```
40G          40행으로 이동한다
```

그리고 파일의 시작점으로 이동하기 위해서는 1G를 사용한다.

1G　　　　　　1행으로 간다

간단히 다음의 지령을 사용하여 파일의 끝으로 갈수 있다.

G　　　　　　파일의 끝으로 간다

주의할것은 파일의 끝으로 이동하려면 행번호를 앞에 놓지 말아야 한다는것이다. 1G와 G는 사용자가 즉시에 기억기에 의뢰하여야 할 지령들이다. 우리가 지금까지 배운 항행지령들을 표 4-5에 묶어 준다.

표 4-5.　　　　vi에서의 항행지령들

지 령	기 능
b	단어의 시작점으로 후진한다
4b	단어의 시작점으로 4단어만큼 후진한다
e	단어의 끝으로 전진한다
w	단어의 시작점으로 전진한다
8w	8번째 단어의 시작점으로 전진한다
0 또는 |	행의 시작점으로 이동한다
30|	30렬에로 이동한다
^	행의 첫 단어에로 이동한다
$	행의 끝으로 이동한다
1G	파일의 시작점으로 이동한다
40G	40행에로 이동한다
G	파일의 끝으로 이동한다

참고

최종행방식은 행들사이의 이동을 위한 동등한 지령들을 제공한다. 앞에서 본 3개의 지령들은 각각 :40, :1, :$로 바뀔수 있다.

참고

만일 파일을 시작할 때 유표가 일정한 행번호에 위치하도록 할 필요가 있다면 그 행번호앞에 +를 붙여 vi의 인수로 사용한다. +만을 사용하면 파일끝으로 이동할것이다.

vi +40 chap01　　　　유표가 40번째 행에 위치한다
vi + chap01　　　　유표가 마지막행에 위치한다

4.9 연산자

본문의 삭제나 복사와 같은 기초편집기능들에 대한 설명으로 넘어 가기전에 우리는 vi가 지원하고 있는 대단히 강력한 특징 즉 **연산자**(operator)에 대하여 리해하여야 한다. vi는 지령방식에서 동작하는 여러개의 한 문자지령들을 가지고 있으며 대부분의 실지편집작업(완충기를 변경시키는 작용들)은 이 연산자들의 사용을 요구한다.

연산자들은 불과 몇개 안되며 사용자는 적어도 아래의 연산자들은 알고 있어야 한다.

d - 삭제

y - 복사(yank)

c - 변경

! - 본문에 작용하는 려파기

연산자는 단독으로는 그 어떤 기능도 수행할수 없으며 어떤 지령과 결합되여야 한다. vi의 능력은 주로 사용자가 이 연산자들을 사용하여 지령들을 조립할수 있다는 편리성에 기인된다. 이것은 사용자가 작업할수 있는 지령들의 새로운 령역을 개설한다. 연산자-지령결합을 사용하여 사용자는 임의의 실천적 환경을 위한 삭제 또는 복사지령을 조립할수 있다. 우리는 다음절에서 연산자들을 폭 넓게 사용하게 될것이다.

표 4-6에는 이 연산자들을 다른 지령방식편집지령들과 결합하여 사용하는 여러가지 가능한 방법들을 보여 준다.

주해

연산자가 자체로서 반복되였을 때(dd, cc, yy, !!와 같이)에는 오직 현재행에서만 작용한다.

표 4-6. vi에서 편집기능들

지 령	기 능
x	유표아래의 한 문자를 지운다
6x	유표아래의 문자와 오른쪽의 5개 문자를 지운다
X	앞문자를 지운다
J	현재행을 다음행과 련결한다
d$ 또는 D	유표로부터 행의 끝까지를 지운다
dd	현재행을 지운다
5dd	5개 행을 지운다
4dw 또는 d4w	4개 단어를 지운다
d30G	유표로부터 행번호 30까지를 지운다
df.	유표로부터 점이 처음으로 나타나는 곳까지를 지운다
d/endif	전진방향에서 유표로부터 처음으로 나타나는 endif문자렬까지를 지운다
yy 또는 Y	현재행을 복사한다(범위에서 Y는 D나 C와 다르다)
y$	유표로부터 행의 끝까지를 복사한다
6yy	6개의 행을 복사한다
yw	현재단어를 복사한다
3yw 또는 y3w	3개 단어를 복사한다
y?case	후진방향에서 유표로부터 처음으로 나타나는 case문자렬까지를 지운다
p	삭제 또는 복사된 본문을 현재행의 아래에 또는 유표의 오른쪽에 놓는다
P	삭제 또는 복사된 본문을 현재행의 우에 또는 유표의 왼쪽에 놓는다
c0	유표로부터 행의 시작까지를 변경한다
c$ 또는 C	유표로부터 행의 끝까지를 변경한다
3cw 또는 c3w	3개 단어를 변경시킨다
cc	현재행을 변경시킨다
cG	유표로부터 파일의 끝까지를 변경시킨다
~	유표아래문자의 크기를 바꾼다
40~	40개 문자의 크기를 바꾼다
.	마지막편집지령을 반복한다
u	마지막편집지령을 취소한다(Linux에서는 종전의 지령)
U	현재행에 만들어 진 모든 변화를 취소한다
[Ctrl-r]	종전의 취소동작을 재실행한다(Linux에서만)

4.10 본문의 삭제, 이동, 복사

이 절에서는 주로 연산자들의 사용에 대해서 취급한다. 이 연산자들을 사용할 때에는 현재의 편집방식이 지령방식인가를 확인하여야 한다. 본문의 복사는 오직 연산자들에 의해서 가능하지만 삭제를 조종하는데는 두가지 방법이 있다.

- 지령방식지령 x와 X를 사용하기
- d연산자를 지령방식지령과 함께 사용하기

본문이동에는 추가적인 단계가 있는데 그것은 삭제된 본문을 새로운 위치에 놓는것이다. 먼저 우리는 가장 단순한 방법으로 삭제를 조종하는 법을 배울것이다.

4.10.1 문자들의 삭제(x, X)

가장 단순한 본문삭제는 x지령으로 수행한다. 이 지령은 유표아래의 문자를 삭제한다. 삭제하려는 문자에 유표를 이동시키고 x를 누른다.

x	한개 문자를 삭제한다

문자는 시야에서 제거되며 오른쪽의 본문이 그 공간을 채우기 위해 왼쪽으로 밀린다. 반복인자는 여기에도 적용되며 따라서 4x는 현재문자뿐아니라 오른쪽으로부터 3개의 문자를 삭제한다.

이 지령의 독특한 특징은 행의 끝에서 x를 눌렀을 때 대부분의 Windows편집기들(지어 emacs까지도)처럼 다음행을 끌어 올리지 않는다는것이다. 그대신 vi는 다음행을 합치지 않고 왼쪽에 있는 문자들을 삭제하기 시작한다.

유표의 왼쪽에 있는 본문을 삭제하려면 X를 사용한다.

X	왼쪽에 있는 본문을 삭제한다

x와는 달리 X는 유표아래의 문자를 삭제하지 않는다. 행의 끝에서 x를 사용하면 이것은 마치 X와 같이 동작하는데 다만 유표아래의 문자도 삭제한다는것이 다르다. 그림 4-13에 x와 X의 사용에 대하여 보여 준다.

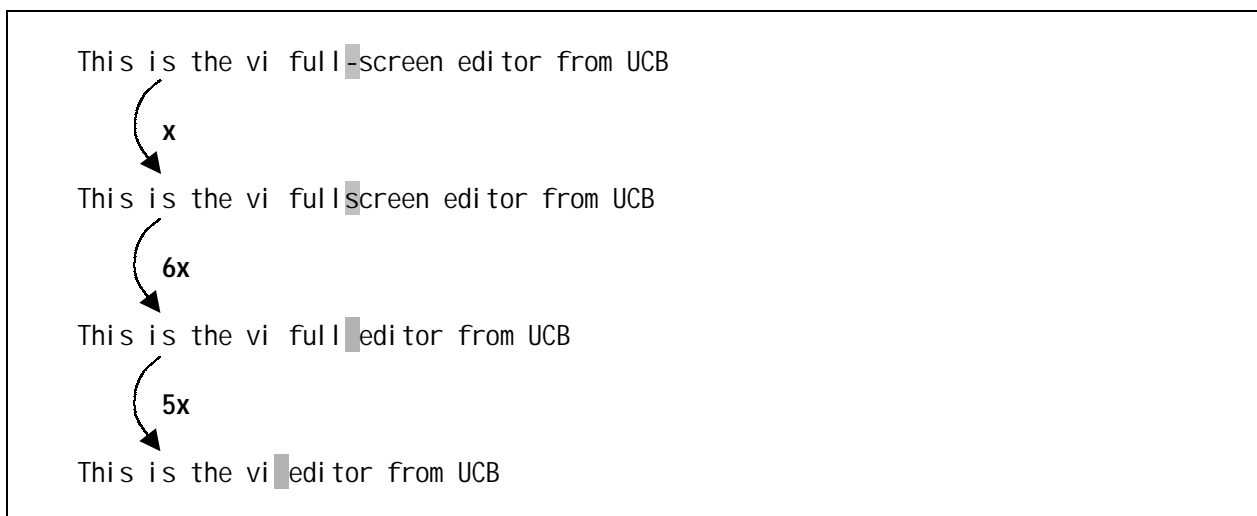



그림 4-13. x와 X지령에 의한 본문삭제

4.10.2 행들의 합치기(J)

방금 우리는 행의 끝에서 삭제건을 눌렀을 때 많은 편집기들이 현재의 행을 다음행과 합친다는데 대하여 언급하였다. vi에서는 J(join)지령으로 동일한 동작을 수행할수 있다. 현재행과 다음행을 합치자면 J를 사용한다.

J　　　　　　4J는 현재행에 다음의 3개 행을 합친다

J는 두 행사이에 있는 **행바꾸기문자**(newline character)를 제거한다(그림 4-14). 그 문자를 위한 ASCII값은 10진수로 10이며 이것을 **LF**(linefeed:행바꾸기)라고 부른다. 그렇지만 행을 합치는것은 편집시에 가질수 있는 행의 최대크기로 제한된다. 초기의 vi들은 1024문자로 제한되였으나 현대체계들은 더 높은 한계를 가지고 있다.

```
if [ $x -gt 0 ] ;
then
        J
if  [ $x -gt 0 ] ; then
```

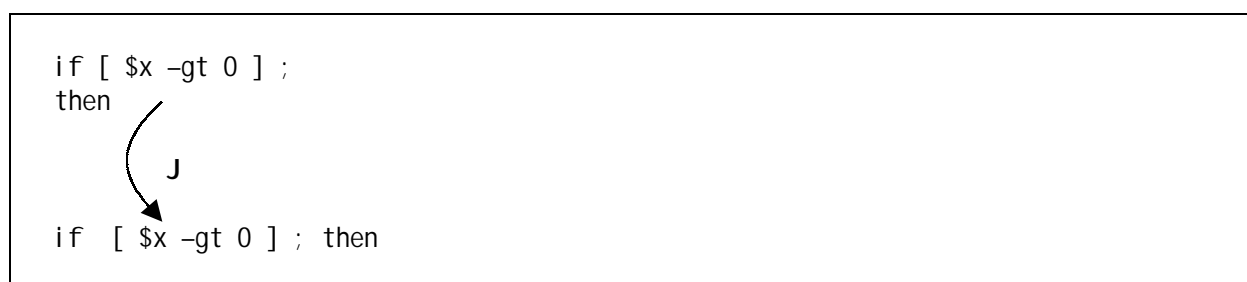



그림 4-14. J지령으로 행들을 합치기

4.10.3 d연산자를 리용한 삭제

현재의 유표위치로부터 행끝까지의 모든 본문을 삭제(delete)하려 한다고 가정하자. 사용자는 행끝으로 이동시키는 초기의 지령을 다시 호출한다면 d연산자를 사용하여 자체로 지령을 조립할수 있다. 이것이 바로 $이며 따라서 그 일감을 수행할 지령은 다음과 같다.

d$　　　　　　행의 끝($)까지 삭제한다

이 지령과 같은 의미를 가진것으로서 D지령이 있다. 마찬가지로 G가 파일끝으로 이행하도록 한다는데로부터 사용자는 다음의 결합을 사용하여 현재의 유표위치로부터 파일끝까지의 모든 본문을 삭제할수 있다.

dG　　　　　　파일끝까지 삭제한다

이와 같은 단순한 론리로부터 우리는 단어를 삭제하는 지령은 dw라고 명백히 말할수 있다. 또한 반복인자도 사용할수 있으며 5dw를 사용하여 5개의 단어들을 삭제할수 있다. 어떻게 특정한 기능들이 단순한 규정묶음에 따라 만들어 질수 있는지 주목할 필요가 있다. 이러한 완성된 선택기능들이 emacs에서는 유용하지 않다.

연산자-지령에 관한 리론은 행들을 삭제할 때에는 뒤자리를 차지하게 된다. 전체 행들은 dd지령(지령이라기보다 2중연산자라고 한다.)을 가지고 삭제한다. 유표를 임의의 행으로 이동시키고 dd를 입력하여 보자.

dd　　　　　　한개 행을 삭제한다

행블로크를 삭제하려면 반복인자를 사용한다. 따라서 현재행과 그아래의 5개의 행들을 삭제하기 위해서는 6dd를 사용한다. 그림 4-15에 행의 삭제와 이동을 위한 d연산자의 사용에 대하여 보여 준다.

```
Press the Esc key for entering command mode
Press the : to enter ex mode
dd is not really a command
It's a combination of an operator and a command

    dd

Press the : to enter ex mode
dd is not really a command
It's a combination of an operator and a command

    2dd

It's a combination of an operator and a command

    P

It's a combination of  an operator and a command
Press the : to enter ex mode
dd is not really a command
```

그림 4-15. dd와 p에 의한 행의 삭제 및 이동

4.10.4 본문의 이동(d, p, P)

본문의 이동은 삭제조작의 확장이다. 즉 사용자는 삭제한 본문을 새 위치에 놓아야 한다. 현재행아래에 전체 행들(dd로 삭제된)을 다시 놓으려면 p를 사용한다. P는 현재행우에 놓을 때 사용된다.

우리가 우에서 잘라 낸 행은 현재행아래에 붙일수 있다. 요구하는 위치로 이동하여 p를 누르자.

```
p            P는 우의 행들을 붙인다
```

p와 P는 단어들을 이동하는데 사용될 때에는 서로 다른 의미들을 가지게 된다. p는 단어들을 왼쪽에, P는 오른쪽에 놓는다. 이 지령들은 또한 본문을 이동시키는데만 그치지 않는다. 복사조작에도 같은 건들이 사용된다.

두 파일사이에서 본문의 이동

우의 기술의 개량된 형식을 사용하면 두 파일들사이에서 잘라붙이기조작을 수행할수 있다. 한 파일로부터 다른 파일에로 한 블로크의 본문을 옮기기 위해서는 다음의 단계들을 거쳐야 한다.

1. 본문을 완충기 a에로 삭제한다. 만일 4개의 행을 삭제한다면 일반삭제지령앞에 문자렬 "a를 붙여 "a4dd를 사용한다.
2. :w로 현재파일을 보관한다.
3. 최종행방식지령 :e foo를 사용하여 새 파일을 연다.
4. 본문을 다시 꺼내기 위하여 요구하는 위치로 조종하여 "와 완충기이름(a), p를 눌러 복사된 본문을 현재행아래에 놓는다("ap).

이 기술은 일반기능의 특수한 경우이다. 여기에 대해서는 4.20에서 설명하였다.

참고

단어들과 행들을 이동하기 위해서는 그것들의 본래의 위치로부터 삭제하여 새로운 위치에 이동한 다음 p 또는 P를 누른다.

4.10.5 y연산자를 사용한 본문의 복사

삭제에 사용된 지령들과 동일한 지령들로 단어들과 행들을 복사한다. 다만 여기서 사용되는 연산자는 d가 아니라 y이다. 이것은 결국 2중연산자 yy가 행을 복사한다는것을 의미한다. 본문의 5개 행을 복사하기 위해서는 유표를 이 행들의 시작에로 이동시킨 다음 5yy를 누른다.

5yy　　　　　원천으로부터 5개 행이 복사된다

그다음 유표를 이 행들이 삽입될 새 위치에로 이동시키고 p를 누른다.

p　　　　　P는 행들을 우에 배치한다

복사된 본문이 현재행의 아래에 배치된다. 같은 론리를 적용하여 우리는 y$가 본문을 현재위치로부터 행끝까지 복사하며 y1G는 현재유표위치로부터 파일시작점까지의 본문을 복사한다고 말할수 있다. p가 복사한 본문을 오른쪽에 놓는가, 아니면 아래에 놓는가는 직접 해보면 알수 있을것이다. 그림 4-16에 행의 복사에 대하여 보여 준다.

```
1. Copying is only possible with the y operator
2. You can use it with any navigation  command
3. y$ copies text from the current position to line end

           First yy then 2j

1. Copying is only possible with the y operator
2. You can use it with any navigation command
3. y$ copies text from the current position to line end

           p

1. Copying is only possible with the y operator
2. You can use it with any navigation commanc
3. y$ copies text from the current position to line end
1. Copying is only possible with the y operator
```

그림 4-16. yy에 의한 행들의 복사

두 파일사이에서 본문의 복사

본문삭제에서 사용한것과 류사한 기술을 리용하여 사용자는 두 파일사이에서 본문을 복사할수 있다. 한 파일로부터 다른 파일로 4개의 단어를 복사하기 위해서는 4dd대신에 4yw를 사용한다. 그밖의 다른것들은 모두 같은데 다만 이번에는 본래의 파일을 보관할 필요가 없을뿐이다.

Linux

본문블로크를 복사 또는 이동하는데서 vim에서는 "a기호들을 전혀 사용할 필요가 없다. 본문을 삭제 또는 복사하고 필요하다면 :w로 파일을 보관하며 :e foo지령을 써서 다음파일에로 절환한 다음 p로 그 본문을 붙인다. vim은 여러 부분이 복사 또는 이동될 때에만 완충기부호들을 요구한다.

4.11 본문의 변경(c, ~)

vi는 본문변경을 위한 몇가지 기술을 제공한다. c연산자를 리용하여 사용자는 우에서 d와 y를 사용한 방법대로 선택을 세밀조종할수 있다. 사용자는 또한 본문의 대소문자를 변경시킬수도 있다. 이 기술들에 대해서는 다음번에 취급한다.

4.11.1 c연산자를 리용한 변경

d와 y를 사용한 동일한 기술을 리용하여 c(change)연산자를 가지고 본문을 변경시킬수 있다. dw가 단어를 삭제한다면 cw는 단어를 변경시킨다. 이것은 입력방식에서 동작하는 유일한 연산자이다. 그것은 사용자가 [Esc]를 써서 변경조작을 완료해야 한다는것을 의미한다.

사용자는 c를 유효한 지령방식지령과 함께 사용하여 입력방식으로 옮겨 갈수 있다. 그러나 이번에는 $가 사용자의 조작범위를 한정하는 경계선을 가리킨다. 만일 사용자가 3cw를 리용하여 3개의 단어들을 변경시킨다면 $는 그 3번째 단어의 끝에 나타나야 한다(Linux는 제외). 삽입된 본문은 화면에 일시적으로 나타난 $가 한정하는 문자들에 덧씌여 진다. 만일 치환된 본문이 더 큰 경우에는 유표가 $표식까지 한번 이동해 간 다음 삽입에 의해 현존본문이 오른쪽으로 밀려 난다.

현재의 유표위치로부터 행끝까지의 본문을 변경시키자면 c$ 또는 C를 사용하며 행전체를 변경시키려면 cc를 사용한다.

```
c$ 또는 C
cc              S지령처럼 동작한다
```

주해 c연산자로 행의 일부 본문을 변경시킬 때 모든 현존본문이 변경될수 있는 곳에 $가 나타나는것을 발견하게 될것이다. $의 오른쪽에 있는 본문은 덧씌여 지지 않고 오른쪽으로 계속 밀려 나갈수 있다. 대부분의 체계들은 $를 나타내지 않고 단순히 변경지령이 입력되는 순간에 정의된 본문을 제거한다.

4.11.2 대소문자의 변경(´)

vi는 ~(물결표)를 사용하여 본문의 대소문자를 반전시킨다. 본문의 한 부분에서 대소문자를 뒤집으려면 유표를 그 부분의 첫 문자에 옮겨 놓고 ~를 누른다.

```
~
```

대소문자가 바뀌여 대문자는 소문자로 되고 소문자는 대문자로 된다. 이 건을 계속 누르고 있으면 유표가 앞으로 전진하면서 맞다들리는 모든 본문들의 대소문자들을 련달아 뒤집는다. 만일 사용자가 100개의 문자들의 렬을 상대로 이러한 작업을 해야 하는 경우에는 반복인자를 사용할수 있다(100~). 그림 4-17에 대소문자의 변환을 보여 준다.

```
Is this a dagger that I see before my eyes?

   16~

iS THIS A DAGGER that I see before my eyes?
```

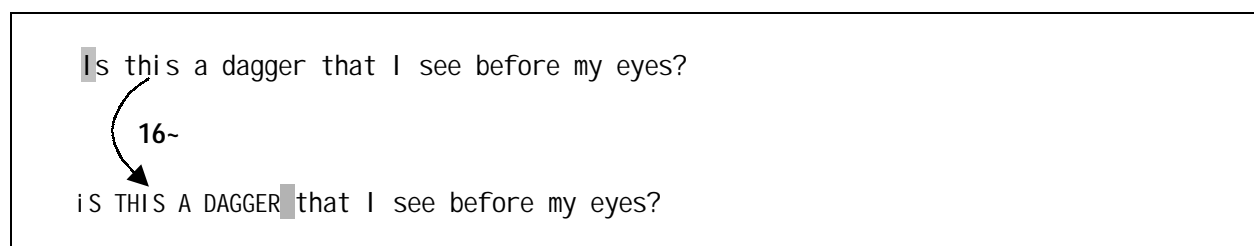



그림 4-17. ~를 리용한 대소문자의 변경

행의 일부 문자들이 이미 대문자로 되여 있는 경우에 그 행전체를 대문자로 변경시키는데는 ~가 적합하지 않다(그 문자들은 반대로 소문자로 전환된다). 완전무결한 대소문자변환을 조종하려면 !연산자를 UNIX의 tr지령과 함께 사용하여야 한다.

4.12 마지막지령의 반복(.)

대부분의 편집기(emacs를 포함한)들은 마지막편집지시를 반복하는 기능을 가지고 있지 않지만 vi는 가지고 있다. 반복은 입력방식지령들과 지령방식지령들에 다 적용된다. 실례로 만일 사용자가 여러 곳에 같은 본문을 삽입하거나 같은 수의 행들을 삭제하려고 한다면 실제적인 삽입 및 삭제지령을 오직 한번만 사용해야 한다. 다른곳에서 그 지령을 반복하자면 사용자는 간단히 .(점)지령을 사용해야 한다. more도 동일한 지령을 사용한다.

간단한 실례를 들자. 만일 사용자가 2dd로 본문의 2개의 행을 삭제하고 다른 곳에서 이 조작을 반복하려고 한다면 사용자가 해야 할 일은 유표를 요구하는 위치에 놓고 .을 누르는것이 전부이다.

.	점

이것은 수행된 마지막편집지시를 반복하며 본문의 2개 행도 역시 삭제될것이다.

일단 입력방식과 지령방식들의 여러가지 지령들에 정통하기만 하면 사용자는 .지령이 대단히 편리하다는것을 알게 될것이다. 실례로 I지령을 사용하여 어떤 행의 시작점에 DROP TABLE이라는 단어들을 삽입하는 경우 사용자는 다음행으로 이동하여 점을 누르는것으로써 그것을 변경시킬수 있다.

주해

점(.)지령은 오직 가장 최근에 진행한 편집조작만을 반복한다(완충기를 변경시킨것). 명백하게 이것은 완충기내용을 변경하지 않는 항행지령들과 탐색지령들은 제외한다. 더우기 탐색지령들은 반복탐색을 위한 자체의 문자묶음을 가지고 있다. 점(.)지령은 또한 최종행방식지령을 반복할수 없다.

4.13 마지막편집지령의 취소(u, U)

만일 편집시에 사용자가 부주의로 하지 말아야 할 수정을 하였다면 vi는 마지막으로 변경된것을 취소(undo)하기 위한 u지령을 제공한다. 사용자는 어떤 작업을 하기전에 u를 눌러야 한다.

u	반복인자는 허용되지 않는다

만일 사용자가 입력방식에 있었다면 u를 사용하기전에 먼저 [Esc]를 누른다. 이 기능은 특히 초학자들에게 매우 유용한데 그 리유는 초학자들이 부주의로 행블로크를 삭제할수도 있기때문이다. 만일 초학자들에게 이와 같은 현상이 생기게 되면 새로운 편집동작이 수행되기전에 즉시 u를 사용해야 한다. 이 단계에서 또 한번 u를 누르면 본래상태를 회복한다.

한개의 행에서 여러개의 수정편집이 진행되였을 때 vi는 사용자가 그 행을 떠나기전에 모든 변경내용들을 포기할수 있게 한다. 다음의 지령은 현재행에서 수행된 모든 변경을 뒤집는다. 즉 유표가 이 행에 이동해 온 때부터 이루어 진 모든 변경이 취소된다.

U	반복인자는 허용되지 않는다

주의

U를 불러 내기전에는 유표가 다른 행으로 이동하지 않았는가를 확인하여야 한다. 유표가 이동한 경우에는 그 지령이 동작하지 않는다. Linux에서는 이 제한이 적용되지 않는다.

Linux

vim은 사용자가 여러개의 편집지령들을 취소하거나 재실행(redo)하게 한다. 여기서 u는 다른 역할을 수행한다. u는 단순히 이전에 진행되였던 작업을 취소한다. 일부 작용들을 재실행하려면 다음의것을 사용하여야 한다.

```
[Ctrl-r]
```

u와 [Ctrl-r]는 다같이 반복인자와 함께 동작한다. 즉 5u는 마지막 5개의 편집동작들을 취소할것이며 3[Ctrl-r]는 마지막 3개의 취소된 지시들을 재실행하게 된다.

vim에서는 U의 기능도 다르다. 유표를 이동시켰다고 해도 현재행에 만들어 진 모든 변경을 취소할수 있다. 하지만 유표를 이동시킨후에 어떤 편집동작을 수행하지 않았는가를 확인해야 한다.

이러한 방법으로 취소시킬수 있는 동작의 회수는 외부적으로 설정될수 있다(undolevels 선택항목으로). 그러나 보통 기정설정(1000)이 적합하다.

4.14 문자렬탐색

vi의 탐색능력은 vi의 강력한 특징들중의 하나이며 emacs외의 임의의 다른 편집기들과 동일하지 않다. 그것은 아주 훌륭하면서도 사용하기 쉽다. vi는 두가지 형식의 탐색을 지원한다.

- 전체 파일에서 문자렬이나 정규식의 탐색
- 현재행에서 한개 문자의 탐색

이 두가지 탐색은 두 방향(앞방향과 뒤방향)에서 탐색을 반복하도록 한다. 표 4-7에 탐색지령과 반복지령들을 보여 준다. 이 절에서는 단순한 문자렬에 의한 탐색을 설명한다. 다음절에서는 정규식들에 대하여 취급하며 일반화된 패턴들이 어떻게 탐색능력을 강화하는가에 대하여 보게 될것이다.

표 4-7. vi에서 지령들의 탐색 및 치환

지 령	기 능
/pat	패턴 pat를 전진탐색한다
?pat	패턴 pat를 후진탐색한다
n	종전의 탐색과 같은 방향에서 탐색을 반복한다(반복인자는 없다)
N	종전의 탐색과 반대방향에서 탐색을 반복한다(반복인자는 없다)
fch	현재행에서 전진방향으로 처음 나타나는 ch문자에로 유표를 이동한다
Fch	우와 같으나 뒤로 이동한다
tch	현재행에서 전진방향으로 처음 나타나는 ch문자의 바로 앞에 유표를 이동한다
Tch	우와 같으나 뒤로 이동한다
;	f, F, t, T로 진행한 탐색을 같은 방향에서 반복한다
,	f, F, t, T로 진행한 탐색을 반대방향에서 반복한다
:n1,n2s/s1/s2/	행 n1부터 n2사이에서 처음 발생하는 문자렬이나 정규식 s1을 문자렬 s2로 교체한다
:1,10s/find/look/g	1행부터 10행까지사이에서 발생하는 모든 find를 look로 교체한다
:.,$s/find/look/gc	현재행으로부터 끝까지에서 대화식으로 find를 look로 교체한다
:s	현재행에서 마지막치환을 반복한다(Linux에서만)

4.14.1 패턴탐색(/, ?)

사용자는 /을 문자렬이나 표현앞에 붙여서 패턴(문자렬 또는 정규식)을 탐색할수 있다. 문자렬 UNIX의 위치를 알아 내기 위해서는 간단히 다음의 지령을 입력한다.

```
/UNIX[Enter]            문자렬 UNIX를 전진탐색한다
```

/부호를 누르면 그것이 화면의 마지막행에 표시된다. 그다음 패턴을 입력하고 [Enter]를 누른다. 그러면 유표는 현재위치로부터 출발하여 처음에 나타나는 패턴의 첫 문자에 위치한다.

만일 패턴이 파일의 끝까지 배치될수 없다면 기정적으로 그 탐색은 본문의 끝을 감돌아 파일의 시작점으로부터 다시 시작된다.

반대방향으로 탐색할 때에는 / 대신에 ?를 사용해야 한다. 다음의 지령은 바로 우에 있는 UNIX를 후진탐색한다.

```
?UNIX[Enter]            뒤로 탐색한다
```

감돌기(wraparound)기능은 여기에도 적용되며 방향은 반대이다. 어느 경우에나 패턴탐색이 실패하는 경우에는 통보문 Pattern not found가 화면의 마지막행에 나타난다.

4.14.2 마지막패턴탐색의 반복(n과 N)

탐색들의 반복을 위해 vi는 n과 N지령들을 사용한다. / 또는 ?로 지정된 종전의 탐색방향으로 탐색을 반복하기 위해서는 n을 사용하여야 한다.

```
n               반드시 전진인것은 아니다
```

반대방향으로 탐색을 반복하기 위하여서는 N을 사용한다.

```
N               반드시 후진인것은 아니다
```

유표는 패턴을 형성하는 문자의 시작점에 놓이게 될것이다. 탐색과정에 파일의 끝이나 시작점에 가닿게 되는 경우에는 감돌기기능이 파일전체가 주사되였음을 확인한다. 지금까지 설명한 탐색지령들과 반복지령들을 그림 4-18에 보여 준다.

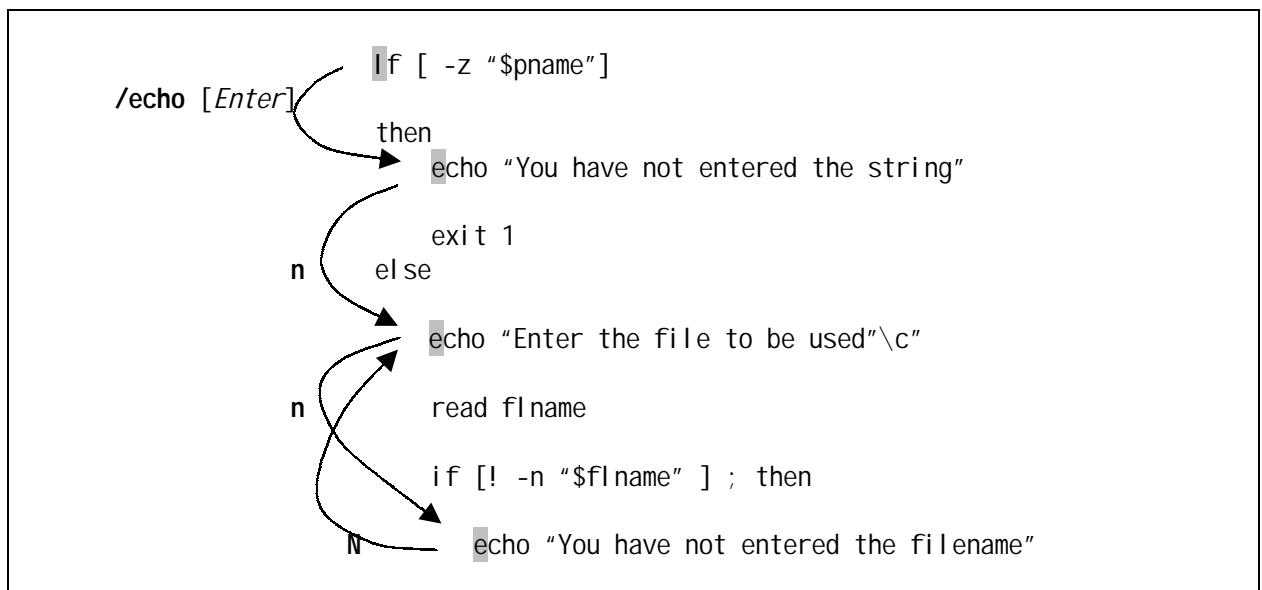



그림 4-18. 패턴탐색과 탐색의 반복

주해 /은 전진방향으로 탐색하고 ?는 반대방향으로 탐색한다. n은 이전 탐색과 같은 방향으로 탐색을 반복한다(반드시 전진방향인것은 아니다). 탐색이 ?로 수행된 경우에 N이 아니라 n이 이 탐색을 반복한다.

사용자는 패턴을 지정하는것으로 vi를 시작할수 있다. 이때 패턴앞에 +/부호를 붙인다.

```
vi +/scheduling chap01          유표는 scheduling에 위치한다
```

참고 유표는 패턴의 첫 실체에 놓이게 될것이다. 사용자는 n과 N을 일상적인 방법으로 사용하여 그 문자렬의 다음번 실체에 유표를 배치할수 있다. 패턴이 많은 단어들을 포함하고 있는 경우에는 인용부호안에 그것들을 넣어 주어야 한다.

4.14.3 문자탐색(f, F, t, T, ;과 ,)

vi는 한 문자지령(f와 F)들을 사용하여 행안의 특정한 문자에로 이동한다. 이 지령들의 뒤에는 탐색하려는 문자가 놓인다. 실례로 문자 (의 다음실체의 위치를 알아 내려면 다음의 지령을 사용한다.

```
f(              후진탐색을 위해서는 F를 사용한다
```

이 탐색은 오직 현재행에만 국한되며 지령 ;(반두점)과 ,(반점)을 사용하여 탐색을 반복할수 있다. 현재행에서 종전의 탐색을 반복하려면 ;지령을 사용한다.

```
;               ,을 사용하여 반대로 탐색한다
```

t와 T는 류사한 기능을 수행하는데 다만 목표했던 문자의 바로 앞에 멈추어 서게 된다. 이 모든 한 문자탐색지령들과 반복지령들도 반복인자와 함께 동작할수 있다.

4.15 정규식을 리용한 탐색

문자렬탐색에는 별로 특수한것이 없으며 모든 편집기가 이 기능을 다 가지고 있다. vi도 탐색표현으로서 일부 특수한 문자들 즉 **메타문자**(metacharacter)들을 가진 일반화된 패턴을 접수한다. 이 패턴을 **정규식**(regular expression)이라고 부르며 류사한 문자렬들의 묶음을 정합하는데 사용된다. 정규식에 대하여서는 후에 풍부히 서술하겠지만 이 표현들에 대한 기초적인 내용들을 모르고서는 탐색에 대한 설명을 원만히 진행할수 없다.

사용자가 지금 michael을 탐색하려고 하는데 파일에서 그 단어의 글자들이 어떻게 씌여 져 있는지 모른다고 하자. michael 혹은 michel이라고 되여 있을수 있다. 두 문자렬에 대하여 각각 탐색하는것보다 하나 또는 그이상의 메타문자들을 사용하는 하나의 표현으로 탐색할수 있다. 이 문자들의 의미에 대해서는 표 4-8에서 보여 준다.

별표(*)

정규식에 의해 자주 사용되는 메타문자들중 하나가 *이다. 두개의 이름 michael과 michel을 참고로 하여 *의 의미를 리해하기로 하자. 하나의 표현을 가지고 두 이름의 위치를 찾으려 한다면 a가 령 또는 그이상 발생한다는것을 가리키도록 *를 a와 함께 사용하여야 한다. 그 패턴안에 a가 있을수도 있고 없을수도 있으므로 이것은 아주 명백하다. 이것은 다음의 지령을 사용하여 두개의 이름들을 찾아 낼수 있다는것을 의미한다.

표 4-8. vi가 사용하는 정규식문자들

기 호	의 미
*	이전 문자의 령이상의 발생과 일치한다
[pqr]	p, q, r중의 어느 한 문자와 일치한다
[^pqr]	p, q, r가 아닌 한 문자와 일치한다
.	한개의 문자와 일치한다
^pat	행의 시작에 있는 패턴 pat와 일치한다
pat$	행의 끝에 있는 패턴 pat와 일치한다
\<pat	단어의 시작에 있는 패턴 pat와 일치한다
pat\>	단어의 끝에 있는 패턴 pat와 일치한다

/micha*el[Enter]   a는 전혀 생기지 않을수도 있다

패턴 micha*el를 정규식이라고 부른다. 이제는 다른 정규식문자들에 주의를 돌려 보자.

문자모임([])

이름 christie와 christy를 고찰하여 보자. 여기서 7번째 문자는 i 또는 y이다. 즉 그것은 문자모임 [iy]에 속하게 된다. 마지막문자 e는 전혀 나타나지 않을수도 있으므로 역시 *를 사용하여 아래와 같이 표현을 조립할수 있다.

/christ[iy]e*   [iy]가 i 또는 y를 정합한다

이것도 두 문자렬의 위치를 얻는다. 문자모임(character class)은 또한 대문자나 소문자로 되여 있을수 있는 문자렬들을 정합하는데도 쓸모가 있다. 아래의 패턴은 strong과 STRONG의 위치를 알아 낸다.

/[sS][tT][rR][oO][nN][gG]   여기서 대소문자는 무의미하다

문자모임안에 놓인 ˆ는 우의것을 부정한다. 그것은 ˆ가 문자모임안에 있는것을 제외한 모든 문자에 정합된다는것을 의미한다. 따라서 /[ˆa-zA-Z]는 자모가 아닌 문자를 정합한다. 범위를 지정할 때에는 왼쪽에 있는 문자가 오른쪽에 있는 문자보다 더 낮은 ASCII값을 가져야 한다. 또한 ˆ는 클라스의 시작점에 놓일 때에만 그 클라스를 부정한다.

패턴의 정착(ˆ와 $)

ˆ가 클라스밖에 놓일 때에는 다른 의미를 가진다. $와 함께 쓰일 때에는 그 무엇과도 정합되지 않으며 다만 정합위치만을 지정한다. ˆ는 행의 시작점에 문자렬을 정착(anchor)시키며 $는 끝에서 그것을 정합한다. C프로그람에서 설명문행을 찾아 보면 그 행의 시작점에 #가 있는것을 보게 된다.

/^#   ˆ는 행의 시작점에서 정합한다

한개 문자(.)

점(.)은 한개 문자를 정합한다. 이것은 행의 특정한 렬에 있는 패턴을 찾아 내는데 쓸모가 있다. 만일 행의 3렬에 있는 문자 w를 정합하려 한다면 다음의 패턴을 사용해야 한다.

/^..w   3번째 렬에서 w를 정합한다

단어들의 정합(\<와 \>)

\<와 \>는 각각 단어의 시작점과 끝에서 패턴을 정합하는데 사용된다. 프로그람을 주사할 때 사용자는 endif와 정합되지 않는 모든 if문들을 찾아 낼수 있다.

```
/\<if\>          ifconfig나 endif와 정합하지 않는다
```

왜 \을사용하는가에 대해서는 후에 리해하게 될것이다. 당분간은 <와 >를 사용할 때 필요한것이라고만 간주하자. \>만을 사용하면 printf가 아니라 print가 있는 행들만을 선택할수 있다.

```
/print\>         \>는 단어끝에서 정합한다
```

이 모든것은 vi와 emacs의 탐색기능들을 다른 대부분의 편집기들보다 우월하게 해준다. grep, sed, awk, perl과 같은 많은 UNIX지령들도 정규식을 사용한다. 그와 관련한 내용들은 제15장의 3개 절에서 취급된다.

4.16 탐색과 치환(:s)

vi는 최종행방식의 :s지령을 리용하여 치환(어떤 문자렬이나 표현을 다른것으로 교체하는것)을 수행한다. w에서와 같이 s의 앞에도 주소가 놓일수 있으며 이 경우 치환은 오직 주소화된 행들에만 작용한다. 문자렬 message를 모두 msg로 치환하는 다음의 실례를 보면 그 문법을 리해할수 있다.

```
:l,$s/message/msg/g[Enter]
```

주소 1,$는 파일의 모든 행들을 표현한다. 패턴들은 /들에 의해 분리되며 g는 치환을 전역화(global)한다. 만일 g파라메터가 사용되지 않는다면 매행에서 첫번째로 나타나는 패턴만이 치환되며 나머지들은 그대로 남아 있게 된다.

만일 패턴을 발견할수 없으면 vi는 다음과 같이 응답한다.

```
Substitute pattern match failed
```

사용자는 또한 치환을 적용할 행들의 범위도 선택할수 있다. 다음의 실례들이 주소화를 명백히 보여 준다.

```
:3,10s/msg/message/g             3행부터 10행까지
:$s/msg/message/g                마지막행만
:1,$s/echo/print/                매행에서 처음에 나타나는것을 치환한다
:.,$s/christ[iy]e*/christie/g              여러가지 패턴을 하나로 치환한다
:.s/ *$//g               현재행에서 꼬리부공간들을 제거한다
:l,$s/^#/dnl/            행의 시작점에서만 #를 치환한다
```

마지막 3개의 실례들은 치환될 패턴으로 정규식을 쓸수 있다는것을 보여 준다. 사용자는 한개의 문자렬 christie를 가지고 그 표현에 정합되는 모든 christe들을 치환할수 있다. 마지막 한개의 실례는 행의 끝($)에 있는 령이상의 공간을 없애 버린다(//로 치환한다). 이외에도 많은 가능성들이 있으며 정규식의 사용은 vi의 기능을 실제적으로 강력하게 하여 준다.

때로는 치환될 패턴이 본래의 패턴을 포함할수도 한다. 사용자는 치환문자렬안에 정합패턴을 의미하는 &부호를 사용할수 있다.

```
:l,$s/Pentium/& II/g            Pentium을 Pentium II로 치환한다
:l,$s/[bB][oO][lL][dD]/<&>/g              bold와 BOLD주위에 <와 >를 제공한다
```

첫번째 실례는 &를 사용하지 않고도 수행할수 있겠지만 두번째 실례는 그렇게 할수 없을것이다. 그것은 치환될 패턴안에 정규식을 반복할수 없기때문이다. 여기서는 bold가 <bold>로, BOLD가 <BOLD>로 치환된다.

사용자는 흔히 문자렬치환을 대화식으로 하려고 할수 있다(emacs의 질문치환). 그 경우에는 끝에 c(confirmatory)파라메터를 추가한다. 치환방법에 대해서는 그림 4-19에 보여 주었고 지령들은 표 4-7에 준다.

매행은 차례로 선택되며 탈자기호(^)들이 련이어 다음행에 뒤따른다. 유표는 이 탈자기호의 끝에 놓여 사용자의 응답을 기다린다. y는 치환을 수행하며 임의의 다른 응답은 수행하지 않는다. 이 절차는 정합된 매행들에서 차례로 반복된다.

치환은 또한 sed지령의 중요한 특징으로서 그 지령도 류사한 문법으로 사용한다. 실지 거기서 취급되는 많은 기능들이 vi에도 적용된다. 치환기능에 대해서는 15.11에서 보다 구체적으로 서술한다.

Linux

유용한 두가지 기능

최종행방식지령의 반복

vim의 일반화된 리력(history)기능은 사용자가 현재의 작업에서 사용한 모든 최종행방식지령들을 다시 호출하여 재실행할수 있게 한다. 이것은 사용자가 마지막치환지령을 반복할수 있을뿐아니라 현재의 작업에서 수행한 이전의 모든 치환을 반복할수 있다는것을 의미한다.

실례로 사용자가 :s로 치환을 수행하였고 이제 그것을 반복하려고 한다면 :을 누른 다음 Up건(유표를 우로 올려 보내는 방향건)을 반복하여 눌러서 이전의 최종행방식지령들을 모두 다시 호출한다. 그다음 필요한 지령을 선택하고 [Enter]건을 누른다. 이 기능은 /과 ?로 진행한 탐색지령들에 대해서도 유용하다. /을 누른 다음 Up건을 누르면 마지막탐색지령이 다시 호출될것이다.

치환

Linux의 치환기능은 보다 읽기 쉽고 편리할뿐아니라 UNIX의 치환기능보다 더 위력하다. 치환될 문자들은 반전색으로 보여 주며 사용자는 화면의 마지막행에 있는 아래와 같은 프롬프트에 대답을 입력해야 한다:

```
replace with msg (y/n/a/q/^E/^Y)?
```

y나 n파는 별도로 사용자는 치환처리를 포기(q)하거나 비대화식(a)으로 만드는 선택항목도 가진다.

4.17 다중파일의 처리

vi는 최종행방식을 사용하여 여러개의 파일들과 완충기들을 처리한다. 사용자는 원하는것만큼의 완충기들을 열수 있으며 한 완충기에서 다른 완충기로 절환할수 있다. 두 편집기들에서의 기초적인 파일조종지령들은 이미 알고 있다. 고급한 지령들에 대하여서는 표 4-9에 보여 준다.

표 4-9. vi의 고급한 파일처리지령들

지 령	기 능
:r foo	현재행의 아래에 파일 foo를 읽어 들인다
:r !date	현재행의 아래에 date지령의 출력을 읽어 들인다
:e foo	현재파일의 편집을 중지하고 파일 foo를 편집한다
:e! foo	우와 같으나 현재파일의 변경을 포기한다
:e!	현재파일의 마지막보관판본을 적재한다(MS Windows의 Revert와 같다)
[Ctrl-^]	가장 최근에 편집된 파일에로 돌아 간다(Solaris에서는 [Shift]가 필요하다)
:e#	우와 갈다
:n	다음파일을 편집한다(vi가 여러개의 파일이름과 함께 호출되였을 때)
:rew	파일목록을 되감아 첫 파일의 편집을 시작한다(vi가 여러개의 파일이름과 함께 호출되였을 때)

```
                Before Substitution

Message="has scheduled the $1 command"
^^^^^^y
e_message="are not using the $1 command"
  ^^^^^^n
if grep "$1" $crondir/henry ; then
     echo "henry $message"
                  ^^^^^^y
elif gep "$" $crondir/Julie ; then
     echo "Julie $message"
                 ^^^^^^y
else
     echo "The three users $e_message"
                              ^^^^^^n
fi

                After Substitution

 msg="has scheduled the $1 command"
 e_message="are not using the $1 command"
 if grep "$1" $crondir/henry ; then
    echo "henry $msg"
 elif grep "$" $crondir/Julie ; then
    echo "Julie $msg"
 else
    echo "The three users $e_message"
 fi
```

그림 4-19. 치환(지령:1,$s/message/msg/gc)

4.17.1 파일과 지령출력의 삽입

대부분의 Windows본문편집기들에는 한 파일의 내용을 다른 파일의 일정한 위치에 삽입하기 위한 쉬운 방법이 없다. 사용자는 파일을 열고 편집차림표로부터 Select All과 같은것을 사용한 다음 [Ctrl-c]로 본문을 복사하고 본래의 파일로 절환하여 [Ctrl-v]로 그 내용을 붙여야 한다. 그것을 UNIX표준들에 의하여 수행되는 많은 량의 작업을 포함한다. vi에서는 다른 파일에 전혀 들어 갈 필요가 없으며 오직 그 내용을 현재의 유표위치에 삽입하면 된다.

```
:r note1        파일 note1을 삽입한다
```

사용자는 또한 자기의 파일에 어떤 지령의 출력을 놓을수 있다. 종전과 같이 :r를 사용하되 파일이름을 지정하는 대신에 !와 지령이름을 입력한다.

```
:r !date        date지령의 출력을 삽입한다
```

이것은 문서작성자들에게 있어서 매우 쓸모 있는 기능이다. 그들은 본문에 지령출력들을 삽입할 필요가 있다. 이 기능을 리용하면 이 출력을 파일에 보존하고 그 파일을 읽어 들일 필요가 없다.

4.17.2 파일절환

때때로 사용자는 자기가 진행한 모든 보관되지 않은 변경내용들을 무시해야 할 때가 있다. 사용자는

마지막으로 보관된 판본을 다시 읽어 들일수 있다.

```
:e!
```

사용자는 또한 편집기를 완료함이 없이 여러개의 파일들을 편집할수 있다. 사용자는 한 파일을 편집하다가 쉽게 다른 파일로 절환할수 있다.

```
:e note2        note2로 절환한다
```

사용자는 다음의 지령들중 어느 하나를 사용하여 본래의 파일로 돌아 올수 있다.

```
[Ctrl-^]        가장 최근에 편집된 파일로 돌아 온다
:e#
```

이것은 **반전효과**(toggling effect)를 가진다. 사용자가 다음번에 우의 지령들중 하나를 사용하면 다시 그 파일로 바뀌여 진다. 첫 지령이 실행하기가 더 쉬우며 여러개의 파일들을 가지고 vi를 사용할 때에 그것을 사용하게 될것이다.

vi를 여러개의 파일이름들과 함께 사용하기

vi가 지령행에서 여러개의 파일이름들과 사용될 때에는 첫번째 파일을 완충기에 읽어 들인다. 사용자는 다음파일로 이동하거나 파일목록을 되풀이하여 처음부터 시작할수 있다.

```
:n              지령행의 다음파일
:rew            지령행의 첫번째 파일로 돌아 간다
```

vi의 보호기구는 현재파일의 변경된 내용이 보관되지 않은 상태에서는 다른 파일로 절환하는것을 막는다. !는 **만능무시스위치**(universal overriding switch)이다. 변경된 내용을 무시하려고 할 때에는 !를 임의의 최종행방식지령과 함께 자유롭게 사용할수 있다(:q!나 :e!로 했던것과 같이). :n!와 :rew!도 같은 방법으로 사용할수 있다.

참고

사용자는 종종 한 파일로부터 다른 파일로 본문을 복사하거나 이동시킬 필요가 제기될수 있다. 다른 파일로 절환하기전에 현재의 파일을 보관시키는 일은 흔히 시끄러운 일로 된다. 절환시에 파일들이 자동보관되게 하려면 파일 .exrc에 :set autowrite 또는 :set aw를 설정해야 한다. 이것은 파일을 특별히 보관하지 않고도 :n을 사용하든가 [Ctrl-^]로 두 파일사이에 절환하는것을 가능하게 할것이다. 그러나 UNIX의 많은 판본들은 파일을 명백히 보관하지 않고 :e foo를 사용하는것을 허용하지 않는다.

지금까지 보여 준 vi의 기능들은 초학자에게 매우 유익하다. 초학자들은 이 기능들을 거의다 정통하기전에는 전진할수 없을것이다.

Linux

창문의 분할

vim에서는 화면을 여러개의 창문들로 가를수 있다. 창문은 비여 있을수도 있고 파일을 가지고 있을수도 있으며 지어 같은 파일의 복사본을 가지고 있을수도 있다. 우리는 여기서 오직 2개의 창문에 대해서만 론의하려고 한다. 왜냐하면 그이상의 창문에서의 작업은 어느정도 불편하기때문이다.

동일한 파일을 2개의 분리된 창문들에서 보려면 최종행방식에서 :sp(split)지령을 사용한다.

```
:sp             현재의 창문을 두개로 가른다
```

갈라 진 창문을 그림 4-20에 보여 준다. 이러한 상태에서 한 창문의 완충기에 만들어 진 변경은 다른 창문에서도 보인다. 사용자는 또한 그 어떤 파일과도 련관되지 않은 새로운 창

문을 만들수도 있다.

:new 새로운 빈 창문

어느 경우에나 사용자는 두개로 갈라 진 화면을 보게 될것이다. 사용자는 아래의 지령을 사용하여 두 창문들사이로 이동할수 있다.

[Ctrl-w][Ctrl-w] 창문들사이로 왕복한다

임의의 창문(비여 있는 혹은 다른 경우)에서 사용자는 :e filename을 사용하여 새로운 파일을 열수 있다. 사용자는 또한 창문의 수직크기를 늘이거나 줄일수도 있다.

[Ctrl-w]+ 현재의 창문크기를 늘인다
[Ctrl-w]- 현재의 창문크기를 줄인다

현재의 창문만을 화면에 남겨 두고 나머지 다른 창문들을 닫아 버리자면 다음의 지령을 사용한다.

:on 다른 모든 창문들을 제거한다

사용자는 현재의 창문을 제거하고 다른 창문으로 이동할수 있다.

:q

사용자가 화면에 여러개의 창문들을 가지고 있을 때 :q는 현재창문에서의 편집에서 탈퇴하고 그것을 닫을것이다. vim에서 모든 창문들에 보관, 탈퇴(quit), 완료(exit)지령들을 적용하려면 현존지령들에 a를 추가해야 한다. 편집내용을 보관하지 않고 모든 창문들에서 탈퇴하려면 :qa를 사용한다. 모든 완충기들을 보관하고 탈퇴하려면 :xa를 사용한다.

```
cd $HOME/project3
echo -e"\
1. Complete backup
2. Incremental backup
3. Complete restore
4. Incremental restore

backup.sh
cd $HOME/project3
echo -e"\
1. Complete backup
2. Incremental backup
3. Complete restore
4. Incremental restore

backup.sh
:sp
```

그림 4-20. vim에서의 갈라 진 창문(File:backup.sh)

4.18 본문에 표식달기

사용자는 파일의 여러 위치들에 표식을 주고 후에 그것들을 찾아 낼수 있다. 열람프로그람들이 제공하는 **서표**(bookmark)와는 달리 이 표식들은 눈으로 볼수 없으며 편집기에서 탈퇴할 때 사라지게 된다. 표식(mark)지령이나 위치찾기지령들은 다같이 위치를 표식하거나 유일적인 표식을 호출하기 위한 문자

와 함께 사용되여야 한다. 어떤 위치를 문자 q로 표식하기 위하여서는 유표를 필요한 위치에 옮기고 다음의 건들을 누른다.

```
mq              표식 q를 만든다
```

사용자는 표식한 본문위치로부터 유표를 다른 곳으로 이동시킨후에 다음의 건들을 사용하여 이 표식에 다시 옮겨 올수 있다.

```
'q              표식 q로 이동한다
```

자모로 된 서로 다른 문자들을 사용하여 사용자는 우와 같은 방법으로 파일에 26개까지의 위치들을 표식할수 있다.

참고

'가 두번 중복되게 되면 유표는 자기의 현재위치와 그전의 위치사이에서 이동한다. 만일 먼저 'a로 a표식에 이동하고 그다음 'b로 b표식에 이동하였다면 사용자는 ''(외인용부호 2개)로 a표식에 돌아 올수 있으며 그다음 다시 같은 지령으로 b표식에 이동해 올수 있다. 이것은 또한 유표가 명백하게 행에서 다른데로 이동해 가지 않은 상태에도 적용된다. 만일 사용자가 30G를 리용하여 어떤 행에로 이동한 다음 G로써 파일끝으로 이동하였다면 사용자는 ''(외인용부호 2개)를 사용하여 이 2개의 위치사이로 오갈수 있다.

4.19 !연산자를 리용한 본문의 려과

!연산자를 리용하면 편집기를 벗어 나지 않고 편집창문의 내용을 정돈할수 있다. UNIX지령들의 계렬에는 어떤 지령으로부터 입력을 받고 다른 지령에 출력을 보내는것도 있다. 이러한 지령들을 **려과기**(filter)라고 하며 UNIX에는 많은 려과기들이 있다. 실례로 sort지령은 파일뿐만아니라 지령의 출력으로부터 받은 입력도 정돈할수 있다. 화면본문에 sort지령을 적용하여 이러한 려과특성을 리용하여 보자.

화면상의 본문려과는 아래와 같은 세 단계를 거치게 된다.

1. 작용할 본문의 시작점에로 이동하여 !를 입력한다.
2. G와 같은 임의의 항행(navigation)지령을 사용하여 본문의 다른 끝으로 이동한다.
3. 그 UNIX지령을 입력하여 그 본문상에서 동작시킨다. 화면의 본문이 즉시 변경된다.

실례로 파일의 21번 행으로부터 40번 행까지 정돈하려면 유표를 21번 행의 첫 문자로 이동시킨 다음 (21G를 리용하여) 아래의 건조작을 수행한다.

```
!40Gsort[Enter]
```

두번째 주소(40G)가 입력될 때까지는 화면에 아무것도 나타나지 않는다. 사용자가 단어 sort를 입력하면 마지막행에 다음과 같은것이 보이게 된다.

```
:20,40!sort           사용자는 최종행방식지령을 실행하였다
```

[Enter]건을 누른 다음 지정된 20개의 행들이 정돈되며 그 출력이 현재의 내용과 교체된다. 사용자는 자기도 모르는 사이에 최종행방식지령 :20,40!sort를 실행하였다. 이 지령을 직접 입력할수도 있다. 그 변화를 보관할수도 있고 만일 사용자의 요구대로 정확히 동작하지 않은 경우에는 그것을 취소할수도 있다.

주해

다른 연산자들과 마찬가지로 !가 두번 중복되는 경우에는 그뒤의 지령을 사용하여 현재의 행에서 조작을 수행한다.

Linux

구역에서의 작업

vim은 본문블로크를 **구역**(region)으로서 표식하고 이 구역에서 몇가지 지령들을 사용할 수 있게 한다. 이 목적을 위해 처음에 정의되지 않은 v건이 사용된다. 이 기술은 간단한 건조작으로서는 부분을 정의할수 없을 때 사용된다. 본문은 또한 강조되여 작업하기 더 쉽게 해준다.

사용자가 v를 누르면 vim은 **보임방식**(visual mode)으로 들어 간다. 마지막행에 통보문 --VISUAL--이 나타난다. 사용자가 조종건으로 유표를 이동시킬 때마다 본문은 편리하게 강조된다. 강조된 구역이 사용자의 구역이며 사용자는 이 구역에서 임의의 vi연산자를 사용하여 작업할수 있다. 사용자는 다시 v를 눌러 구역정의를 취소할수도 있다.

만일 구역을 삭제하려면 연산자 d를 누르면 된다. 이때에는 d에 그 어떤 추가적인 지령을 결합할 필요가 없다. y를 사용하여 선택구역을 복사한다. 화면본문을 려파하기 위해서는 !가 앞에 붙은 지령을 사용한다(구역을 정돈하기 위하여 !sort를 사용한것처럼). ~는 비록 연산자는 아니지만 이것도 본문의 대소문자를 변경시키는데 사용할수 있다.

4.20 이름붙은완충기를 리용한 다중본문구역의 복사와 이동

우리는 본문의 한개 구역을 복사 및 이동하였다(지어 두 파일사이에도 이러한 작업이 수행되였다). 한번의 파일절환으로 프로그람의 여러 구역들을 다른 파일에 복사해야 할 필요성을 체험해 본 일이 있을수 있다. vi와 emacs는 다같이 26개까지의 본문블로크(자모문자를 따서 이름 지은)들을 여러개의 특수한 **이름붙은완충기**(named buffer)들에 저장하는 기능을 제공한다.

우리는 이미 두 파일을 가지고 작업할 때 이름붙은완충기를 사용하여 보았다. 그때 우리는 복사조작이나 삭제조작을 하기전에 "a를 사용하였다. a는 vi에서 리용할수 있는 26개 완충기들중의 하나이며 실지로 사용자는 임의의 완충기이름을 선택할수 있다. 아래에 4개의 행을 이름붙은완충기 v에 복사하는 방법을 보여 준다.

"v4yy	4개의 행을 완충기 v에 복사한다

본문을 다시 꺼내려면 요구하는 위치로 이동하여 다음의 지령을 사용한다.

"vp	본문을 우에 놓기 위해서는 P를 사용한다

이러한 방법으로 사용자는 26개까지의 본문구역들을 복사하여 그 전부를 서로 다른 위치에(지어는 서로 다른 파일에) 회복할수 있다. Windows와 그 마우스는 이 모든것을 전혀 할수 없다. 하지만 X Window체계와 그의 xclpboard의뢰기(12.11.1)는 이보다 더 잘할수 있다.

사용자가 이미 일정한 본문을 가지고 있는 이름붙은완충기에서 조작을 수행하면 완충기의 내용이 겹쳐 씌여 진다. vi는 이 완충기들에 추가하기 위하여 대문자를 사용한다("ayy대신에 "Ayy를 리용하여).

참고

만일 서로 다른 위치들에서 본문을 복사하여 그 전량을 한 위치에 넣으려 한다면 사용자는 대문자로 된 완충기이름을 사용하여 그 완충기에 추가하여야 한다. 실례로 사용자는 10개의 위치에서 "Ayy를 반복하여 누름으로써 그 10개의 위치들에서 현재행을 복사할수 있다. 그러면 완충기 A는 10개의 행을 모두 포함하게 되며 "ap를 한번 호출하여 그것들을 넣을수 있다.

4.21 다중삭제의 회복

만일 사용자가 서로 다른 위치에서 여러개의 본문구역을 삭제한 경우 사용자는 완전한 행의 마지막 9개의 삭제들을 회복할수 있다. 완전행의 삭제는 **번호붙은완충기**(1~9)들에 저장된다(행의 부분적인 삭제는 제외). 가장 최근의 삭제는 완충기 1에, 그다음것은 완충기 2에, 이런 식으로 저장된다. 이름붙은완충기들에서와 같이 완충기들에 접근하는데 동일한 접두사 "가 리용된다. 가장 최근에 삭제된 한 묶음의 행들을 회복하려면 다음의 지령을 사용한다.

```
"1p                 가장 최근의 삭제를 회복한다
```

만일 회복된 본문이 사용자가 기대하였던것이 아니라면 u를 리용하여 마지막회복조작을 취소한 다음 "2p를 사용한다. 사용자는 이런 식으로 매번 "와 수자 그리고 p를 리용하는 방법을 쓸수 있다.

더 좋은 방법이 있다. vi는 .지령을 마지막지령을 반복하는 일반적인 기능을 수행하는데뿐아니라 삭제된 행들을 회복하는데도 사용한다. 그 지령은 설정된 완충기를 거쳐 다음완충기의 내용들을 회복하여 나간다.

점(.)을 이와 같은 방법으로 작업시키기 위해서는 "1p를 리용하여 삭제한 내용을 적어도 한번은 회복하여야 한다. 만일 사용자가 찾고 있는 블로크가 아닌 경우에는 u로 취소하고 .을 눌러서 다음완충기(2)를 회복한다. 만일 그 본문이 완충기 4에 존재하고 있다면 사용자는 완충기번호 1로 시작하여 다음과 같은 지령들을 줄수 있다.

```
"1pu.u.u.
```

주해

점(.)은 마지막 9번의 완전행삭제만을 회복할수 있다. 행의 부분들은 이 방법으로 회복될수 없다. vim에는 이러한 제한이 적용되지 않는다.

4.22 본문의 생략(:ab)

사용자는 긴 단어들을 보다 짧은 문자렬들로 생략할수 있다. 사용자가 략어(abbreviation)를 입력하고 뒤따라 공백이나 점문자를 입력하면 즉시 그 문자렬이 확장된다. 문서작성자들과 프로그람작성자들은 자주 이 기능을 리용하여 자주 사용되는 긴 단어들과 타자할 때 자주 틀리군 하는 단어들을 생략한다.

본문을 생략하기 위해서는 최종행방식의 :abbreviate지령(그자체도 :ab로 생략된다.)을 사용해야 한다. 아래에 Java에서 거의나 자주 리용되는 지령을 포함하는 2개의 유용한 략어들을 준다.

```
:ab varaible variable              글자를 바로 잡는다
:ab sopl System.out.println        sopl는 System.out.print로 된다
```

사실 첫번째것은 생략이 아니며 글자가 자주 틀리군 하는 단어들을 바로 잡는데 이 기능을 사용하게 한것이다. 단어 sopl을 입력하고 뒤이어 자모나 수자도 아니고 _(밑줄)문자도 아닌 어떤 건을 누르면 sopl이 System.out.println으로 되는것을 보게 된다.

비록 이 생략지령(abbreviation command)들은 최종행방식에서 사용할수 있는것들이지만 사용자들은 시작파일 .exrc에 그 항목을 배치함으로써 그것들을 영구적인것으로 만들려고 할것이다. 그 항목들은 최종행방식프롬프트에 입력될 때와 꼭 같은 방식으로 거기에 배치되여야 한다. :ab를 리용하여 략어들을 목록화하며 :unab를 써서 취소한다.

주해 시작파일 .exrc는 홈등록부에 배치된다. 사용자는 가입한 즉시 pwd지령을 사용하여 자기의 홈등록부를 알수 있다.

4.23 건반의 전용화(:map)

사용자가 편집기다루기에 숙련되면 몇개의 건누름렬(keystrock sequence)을 반복적으로 실행하여 일감을 수행하는것을 발견하게 될것이다. 이번에는 자주 사용되는 건누름렬을 하나의 건에 대응시켜 보자. 사용자는 정의되지 않은 건들을 할당하거나 정의된 건들을 재할당함으로써 이 건을 눌렀을 때에 그것이 지령렬로 확장되여 그 작업을 하도록 할수 있다.

vi는 건들을 대응시키는데 :map를 사용한다. 그 지령의 뒤에는 대응시킬 건과 대응되여야 할 건누름렬이 놓인다. 실례로 만일 사용자가 자기의 완충기를 한개의 건누름으로 보관하려면 일정한 건(여기서는 q라고 하자)을 아래와 같은 건누름렬에 대응시킬수 있다.

```
:map q :w^M                ^M은 [Enter]건을 의미한다
```

이 대응은 [Enter]건도 포함하는데 vi는 이것을 [Ctrl-m](^M으로 나타낸다.)으로 리해한다. 이 문자는 처음에 [Ctrl-v]를 누르고 그다음 [Ctrl-m]을 눌러서 입력한다. q가 일단 대응되면 사용자는 그것을 눌러 자기의 파일을 보관할수 있다.

사용자는 또한 입력방식에서도 건들을 대응시킬수 있다. 이때에는 map의 다음에 !를 사용하여야 한다. 두번째 기능건 [F2]를 우의 기능에 대응시켜 보자.

```
:map! #2 ^[:w^M            기능건 [F2]는 #2이다
```

처음 대응된 문자는 [Esc]로서 ^[로 표시된다. 사용자는 :w를 사용하기전에 지령방식에로 절환하려면 이 건을 눌러야 한다. 이 대응은 지령방식에서는 동작하지 않는다.

참고 vi에서 탈퇴하지 않고 쉘이나 perl을 실행시킬수 있는가? 이것을 기능건 [F1]에 대응시켜 보자.

```
:map #1 :!%^M              파일은 실행가능한 허가권을 가져야 한다
```

이것은 현재의 파일(%)을 두점(:)프롬프트에서 실행(!)시킨다. 이것은 vi에서 탈퇴하지 않고 스크립트를 실행시키는데 사용되는 하나의 중요한 대응이다. vi는 현재의 파일을 %로서 리해한다. 만일 편집기로부터 현재의 파일을 콤파일하려 한다면 콤파일지령(실례로 cc 또는 javac)에 %를 인수로 사용해야 할것이다.

생략지령들을 리용하려면 .exrc에 유용한 넘기기들을 보관하여야 한다. :map지령은 넘긴 환경을 현시한다. :unmap는 **지령방식넘기기**를 취소하며 :unmap!는 **입력방식넘기기**를 취소한다.

참고 이제는 아마 g, q, v, K, V, Z건들이 vi에서 정의되지 않았다는것을 알수 있을것이다. 이 건들은 건반을 전용화하는데 사용할수 있다. 그러나 vim에서는 이 건들의 일부가 정의되여 있다.

4.24 환경의 전용화(:set)

vi의 동작은 그 변수들의 설정에 의하여 결정된다. 이런 변수들이 많기는 하지만 일반적으로 일상적인 편집작업에는 기정설정이 적합하다. 그러나 때때로 응용프로그람에 편리하게 그 변수들을 변경시켜야

할 때가 있다. 편집기에서 할수도 있지만 그 설정들을 영구화하자면 .exrc파일에 배치하는것이 더 좋다.

vi는 최종행방식의 :set지령을 사용하여 변수들을 설정한다(.exrc안에 배치되는 경우 :은 선택적이다). 대표적인 두개의 설정을 아래에 준다.

```
:set showmode
:set ignorecase          동의어로서 ic도 사용할수 있다
```

두번째 변수는 생략될수 있다. 즉 :set ic를 사용할수도 있다. 모든 변수들의 이름앞에는 문자렬 no가 붙을수 있는데 그 경우에는 설정이 반전되거나 비능동상태로 된다. 실례로 noignorecase는 ignorecase를 부정한다. 이 설정은 :set noic로 생략될수도 있다. 일부 변수들의 의미는 다음과 같다.

showmode: 이 지령은 현재 vi가 위치하고 있는 방식을 알려 준다. 초학자들은 필요없이 자주 건을 사용하려는 경향이 있는데 지령방식이 아니라는것을 나타내는 마지막행의 INSERT나 REPLACE를 보는것이 좋다.

showmatch: 이 설정은 프로그람작성자가)나 }에 대응한 괄호를 즉시에 볼수 있게 한다.)나 }가 입력되는 순간 유표는 그와 대응되는 괄호로 이동하며 현재위치로 돌아 오기전에 거기에 잠간 머무른다. 만일 찾지 못하면 경고음을 낸다.

autowrite: 다중파일편집시에 vi는 현재의 완충기가 보관되지 않았으면 :n이나 [Ctrl-^]을 써도 다른 파일로 절환하지 않게 한다. 이것은 빈번히 사용자를 당황하게 만들며 autowrite설정은 다음파일로 이동하기전에 현재완충기가 자동적으로 보관되게 한다.

우리는 대소문자를 무시하고 탐색을 진행할수 있다(ignorecase). 이것은 단어를 대문자로 썼는지 소문자로 썼는지 모르는 경우에 도움이 된다. 또한 정규식문자들의 뜻을 없애고 문자렬들을 문자그대로 정합할수도 있다(nomagic). 타브문자를 기정적인 8대신에 4로 설정할수 있다(tabstop 4). 또한 오유수정작업을 편리하게 할수 있도록 프로그람에 행번호를 붙일수도 있다(number).

:set all지령을 사용하면 모든 설정들을 현시할수 있다. 일반적인 :set선택항목들을 표 4-10에 보여준다.

표 4-10. vi에서 :set선택항목들

선택항목	략 어	의 미
autoindent	ai	종전의 들여쓰기준위에서 다음행을 시작한다
autowrite	aw	:n이나 [Ctrl-^]를 리용하여 파일을 절환할 때마다 자동적으로 현재파일을 써넣는다
ignorecase	ic	패턴탐색에서 대소문자를 무시한다
magic		패턴탐색시에 정규식묶음의 문자들을 특정한 문자로서 취급한다
number	nu	화면에 행번호를 현시한다
showmatch	sm	)나 }의 짝을 즉시 보여 준다
showmode		vi가 입력방식에 있을 때 통보문을 현시한다
tabstop	ts	현시를 위한 타브를 설정한다(기정값은 8개의 공백문자)
wrapscan	ws	파일전체를 주사하도록 하기 위하여 파일의 다른 끝으로 옮겨 가 패턴탐색을 계속한다

요 약

vi는 3가지 방식으로 동작하며 한 방식에서 다른 방식에로 매우 쉽게 절환할수 있다.

지령방식은 본문을 조작하거나 항행을 조종하는 지령들을 입력하는데 사용된다. vi의 거의 모든 기능들이 이 방식에서 동작한다.

입력방식은 본문의 삽입(i와 I), 추가(a와 A), 변경(s와 S)에 쓰이며 행을 여는데(o와 O)도 리용된다. 이 방식은 [Esc]건을 눌러 완료한다. 먼저 [Ctrl-v]를 누르고 그다음 문자를 누르는 방법으로 조종문자들을 입력할수 있다.

최종행방식은 파일처리와 치환에 사용된다. 지령방식에서 :을 누르면 이 방식이 호출된다.

입력방식과 지령방식의 거의 모든 지령들은 반복인자와 함께 쓰이는데 여기서 반복인자는 그 지령을 여러번 수행하게 한다.

최종행방식을 사용하여 작업의 보관(:w), 보관후 탈퇴(:x와 :wq), 보관없이 탈퇴(:q와 :q!)를 진행할수 있다. 행주소들을 :w와 함께 사용하면 선택된 행들을 별개의 파일에 써넣을수 있다. 이 방식에서 현재행은 점(.)으로, 마지막행은 $로 표현한다.

체계파괴로 하여 잃어 진 보관되지 않은 작업을 살리기 위하여서는 vi를 -r선택항목과 함께 호출하여야 한다.

지령방식에서는 항행단위로 단어를 사용함으로써 행을 따라 이동할수 있다. 단어의 시작 또는 행의 시작(0)과 끝($)으로 후진(b)하거나 전진(w)할수 있다. 현재행번호를 알아 낼수 있으며([Ctrl-g]) 지정된 행번호에로 이행할수 있다(G). 조종건을 사용하여 앞페지 또는 뒤페지로 이동할수 있으며 우로(k), 아래로(j), 오른쪽으로(l), 왼쪽으로(h) 이동할수 있다.

문자를 삭제할수 있으며(x와X) 지령방식지령과 연산자를 리용하여 임의의 방법으로 본문을 삭제(d)하거나 이동 및 복사(y)할수 있다. p나 P를 써서 본문을 새로운 위치에 배치한다. 연산자가 혼자서 리용될 때(dd와 같이) 그 연산자는 현재행에서만 동작한다.

복사조작이나 삭제조작뿐아니라 붙이기조작에서도 앞에 "a기호를 사용하여 한 파일에서 다른 파일로 본문을 복사하거나 옮길수 있다.

본문의 대소문자는 반전될수 있다(~). c연산자를 사용하여 더 융통성 있는 방법으로 본문을 변경할수 있다.

vi는 마지막편집지령을 반복(.)하거나 취소(u)할수 있다. 유표가 다른 행으로 옮겨 가지 않은 상태에서는 현재행에 만들어 진 모든 변경들을 취소(U)할수 있다. Linux에서 vim은 u와 [Ctrl-r]로 여러준위의 취소와 재실행을 각각 수행할수 있다.

패턴에 의한 탐색을 진행할수 있으며(/와 ?) 그 탐색을 두 방향에서 반복할수도 있다(n과 N). 정규식을 사용하면 하나의 표현을 가지고 하나이상의 패턴을 탐색할수 있다. 전역적으로 또는 대화적으로 한 패턴을 다른 문자렬로 치환할수 있다(:s). 또한 행안에서 문자에 대한 탐색도 진행할수 있으며(f) 그 탐색을 반복할수 있다(;).

다른 파일의 내용을 삽입하거나(:r) 다른 파일로 이동하거나(:e) 마지막으로 편집된 파일로 돌아 가거나([Ctrl-^]) 마지막으로 보관된 판본을 되살릴수 있다(:e!).

또한 vi를 여러개의 파일이름들과 함께 호출하여 매 파일을 차례로 편집하며(:n) 파일목록을 처음부터 시작하도록 다시 감을수 있다(:rew). vi에서는 일반적으로 현재파일의 변경이 보관되지 않는 경우에는 뒤붙이 !를 쓰지 않고서는 파일을 바꿀수 없다.

m과 한개의 문자를 사용하여 26개의 위치(a~z)를 표식할수 있으며 '와 그 문자를 사용하여 표식된

위치를 호출할수 있다 .

UNIX지령으로 본문을 려과할수도 있다.

본문을 저장하기 위한 26개까지의 완충기(a~z)들을 사용할수 있다(완충기이름앞에 "가 붙는다). p나 P를 리용하여 본문을 다른 파일에 배치할수 있다.

1~9까지의 수자를 리용하여 9개까지의 완전행삭제를 취소할수 있다. 점(.)이 지워 진 행들을 회복하는데 사용될 때에는 완충기들을 차례로 거치면서 이전의 삭제된 행들을 회복한다.

긴 문자렬들을 짧은 문자렬들로 생략(:ab)할수 있으며 짧은 문자렬을 입력하면 자동적으로 확장된다.

자주 사용하는 지령렬들을 하나의 건에 대응시킬수 있다(지령방식에서는 :map, 입력방식에서는 :map!) [Enter]를 나타내려면 [Ctrl-m]을 사용한다.

:set지령으로는 파일절환전에 파일이 보관되도록 할수 있으며(autowrite) 대소문자를 무시한 탐색을 진행할수 있고(ignorecase) 정규식의 의미를 없앨수 있다(nomagic).

기동해서부터 :ab, :map, :set지령들이 항상 유용하도록 하려면 그 지령들을 .exrc파일에 배치하여야 한다.

시험문제

1. 행의 내용을 부분적으로 덧쓰기하려면 어떻게 해야 하는가?
2. 첫행앞에 한개의 행을 삽입하려면 어떻게 해야 하는가?
3. bad를 bat로 어떻게 변경시키는가?
4. 현재행의 내용들을 어떻게 완전히 변경시키는가?
5. 편집작업을 어떻게 포기하는가?
6. 행의 40번째 문자로 어떻게 옮겨 가는가?
7. 6행우로 이동하려면 어떻게 해야 하는가?
8. 화면이 헝클어 졌다. 그것을 어떻게 지우겠는가?
9. 현재행을 별개의 파일에 어떻게 보관하는가?
10. 10개의 단어를 어떻게 복사하는가?
11. d와 y는 지령방식지령들과 어떻게 다른가?
12. 파일의 모든 행들에서 비대화적으로 그리고 전역적으로 Internet를 Web로 교체하려면 어떻게 해야 하는가?
13. 현재행부터 파일의 시작까지의 본문을 어떻게 지우는가?
14. 파일의 마지막보관판본을 어떻게 되살리는가?
15. vim에서 vi에 비하여 개선된 3개의 주요기능을 말하시오.
16. vim에서 취소했던 조작을 어떻게 재실행시키는가?

련습문제

1. 행의 끝에 */를 어떻게 추가하는가?
2. 최소한의 건조작으로 has를 have로 교체하려면 어떻게 해야 하는가?
3. 편집내용을 보관하고 vi에서 탈퇴하는 3가지 방법을 말하시오.
4. 어떻게 행의 5번째 단어로 재빨리 이동하여 그 단어의 4개의 문자를 임의의 문자로 교체하겠는가?
5. 어떻게 100번째 행으로 이동하여 그 행을 포함한 나머지행들을 개별적인 파일에 쓰겠는가?

6. 작업중에 어떻게 하면 가입한 사용자들의 목록을 볼수 있는가?
7. 아래의 문자렬에 몇개의 단어들이 있는가?

   `02.29.2000 is_last_day_of_February`

8. 다음의 지령들가운데서 어느 지령이 반복되거나 취소될수 있는가를 설명하시오.

   (i) `40k` (ii) `[Ctrl-f]` (iii) `5x` (iv) `J`

9. 틀리게 입력한 단어 Comptuer가 있다. 이것을 어떻게 Computer로 정정하겠는가?
10. 두개의 행을 파일의 시작으로부터 파일의 끝으로 옮기려면 어떻게 해야 하는가?
11. 5개의 행을 어떻게 하나의 행으로 결합하는가?
12. 20yy에 의하여 20개의 행을 복사하고 :e foo에 의하여 다른 파일로 절환한 다음 p로 그 행들을 다시 붙이려 하였지만 되지 않았다. 왜 그런가?
13. 현재위치에서부터 처음 나타나는 Packet Switching이라는 패턴까지의 본문을 변경시키는 지령을 작성하시오.
14. 소문자들로만 이루어 진 행전체를 어떻게 대문자로 변경하는가?
15. 어떻게 문자렬탐색을 반복하는가? 전진탐색으로 하겠는가 후진탐색으로 하겠는가?
16. 첫 다섯개의 행들을 오른쪽으로 두자리 공백을 주어 움직여야 한다. 대화적으로 하려면 어떻게 해야 하는가?
17. 우와 같은 조작을 비대화적으로 하자면 어떻게 해야 하는가?
18. 어떻게 Bill Joy라는 단어를 찾은 다음 William이라는 단어를 매번 입력하지 않고 단어 Bill Joy의 일부인 Bill을 Willam이라고 바꿀것인가?
19. 지령 /*와 f*사이의 차이점은 무엇인가?
20. 행에서 첫 두개의 문장을 지우는 지령을 작성하시오. 다음행들에서 이 조작을 반복하려면 어떻게 하여야 하는가?
21. 파일의 모든 행에서 모든 선두공백들을 어떻게 한자리 공백으로 교체하는가?
22. U지령은 무엇을 하는가? 그 작업은 언제 실패하는가?
23. vi에서 탈퇴하지 않고 어떻게 두번째 파일을 편집하며 두 파일을 어떻게 절환하는가?
24. 어떻게 한 파일에서 다른 파일로 이동할 때 vi가 자동적으로 파일을 보관하게 할것인가?
25. 현존본문의 대소문자크기에 관계없이 행전체를 대문자로 변경하려면 어떻게 해야 하는가? (참고:먼저 9장을 읽으시오)
26. 파일의 시작과 끝사이에서 어떻게 반복적으로 왕복하는가?
27. 10행짜리 묶음과 5행짜리 묶음을 각각 한 파일에서 다른 파일로 어떻게 복사하는가?
28. 부주의로 3개의 행묶음들을 10dd, 5dd, dd로(차례로) 지웠다. 10개의 지워 진 행들을 회복하려면 어떻게 하여야 하는가?
29. regular expression으로 확장되도록 략어 re를 정의하였으며 단어 re-rating을 입력하면 그 방법으로 확장된다. 무엇을 하여야 하는가?
30. 행에서 모든 선두공백을 제거하는 넘기기를 쓰시오.
31. 편집기에서 탈퇴하지 않고 어떻게 현재편집중에 있는 C프로그람을 콤파일할것인가?
32. 쓰이지 않는 임의의 건을 사용하여 다음의 빈 행(공백을 포함할수도 있고 포함하지 않을수도 있다.)을 찾아 내서 제거하는 넘기기를 구성하시오.
33. 점(.)지령의 두가지 기능은 무엇인가?

제 5 장. GNU emacs편집기

UNIX는 vi외에도 또 하나의 강력한 전화면편집기 emacs를 가지고 있다. 이 편집기는 GNU(지금은 자유쏘프트웨어재단이라고 한다.)의 창시자인 리챠드 스톨맨(Richard Stallman)이 만들었다. 이 프로그람은 원래 TECO편집기를 위한 마크로들의 묶음으로서 작성되였지만 독자적인 편집기로 되게 하기 위하여 여러번 다시 작성되였다. vi와 달리 emacs는 모든 UNIX체계들에서 리용할수 있는것은 아니지만 Linux에서는 표준편집기로 되여 있다. 이 장에서는 GNU판본에 대하여 취급한다.

GNU emacs는 편집기로서의 기능외에도 여러가지 비편집기능들을 가진다. 그러나 이 장에서는 편집기능들에 대하여서만 서술한다. vi와 emacs는 둘 다 제나름의 장점(과 약점)을 가지고 있다. vi는 보다 적은 건조작으로 편집을 하지만 emacs는 마크로들을 사용하여 탐색과 치환을 진행하는 측면에서 vi를 압도한다.

또 다른 편집기의 지령들을 배울 필요가 있는가? 이 질문에는 쉽게 대답하지는 못하겠지만 아마 하나를 적당히 안 다음 다른것을 받아 들이는것이 더 좋을것이다. 이 장에서는 종전과 류사한 제목들의 순서로 서술한다. 앞장에서 vi의 주요내용을 수집했다면 이 장에서는 emacs에서 무엇을 기대할수 있는가를 알게 될것이다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- [Ctrl], [Alt], [Esc]건들의 사용법을 배운다(5.1.1).
- [Alt-x]와 함께 emacs지령들을 입력한다(5.1.3).
- 편집작업을 시작하며 편집기에서 탈퇴한다(5.2).
- 본문과 조종문자들을 입력한다(5.3).
- 작업내용을 보관하며 파괴로부터 회복한다(5.4).
- 수자인수의 의미를 리해한다(5.5).
- 파일안에서, 행안에서 유표를 항행시킨다(5.6).
- 구역을 사용하거나 사용함이 없이 본문을 지우고 이동하며 복사한다(5.8).
- 제거고리(kill ring)의 의미를 리해한다(5.8).
- 구역을 정의하거나 정의함이 없이 본문의 대소문자를 변경시킨다(5.9).
- 지령완성하기기능을 리용하여 지령전체를 입력하지 않고 지령을 완성한다(5.10).
- 종전지령들을 취소하거나 재실행시킨다(5.11).
- 문자렬의 증가탐색과 비증가탐색을 수행한다(5.12).
- 질문치환기능을 리용하여 대화적으로, 비대화적으로 문자렬을 치환한다(5.13, 5.14).
- 정규식을 포함하기 위하여 탐색 및 치환기구를 확장한다(5.13, 5.14).
- 두개의 창문을 가지고 작업하며 여러개의 파일을 편집한다(5.15).
- 효과적인 도움말기능을 사용한다(5.17).
- 서표를 리용하거나 리용함이 없이 본문에 표식을 주고 임의의 표식을 호출한다(5.18).
- UNIX지령을 실행(려과)하여 화면상에서 본문을 변경한다(5.19).
- 완충기들을 분할하여 여러 본문단락들을 복사한다(5.20).
- 복사 또는 삭제된 본문을 제거고리로부터 회복한다(5.21).
- 략어들과 마크로들을 정의하고 변수들을 설정하여 편집기의 환경을 전용화한다(5.22부터 5.25까지).

5.1 emacs의 기초

vi와 달리 emacs는 체계에 자동적으로 설치되여 있지 않을수도 있다. 그러므로 emacs가 설치되였는가를 확인하고 파일이름과 함께 호출하여 보자.

```
emacs emfile
```

사용자에게는 또다시 전화면이 펼쳐 지지만 편집에 유용한것은 25개중 22개행뿐이다. 2개의 행은 반전색으로 현시된다. 우의 행은 차림표를 보여 주며 아래의 행은 **방식행**(mode line)을 보여 준다. 이 방식행의 아래에 있는것은 최종행 즉 emacs가 발생시키는 통보문을 보여 주는 **소형완충기**(minibuffer)이다(그림 5-1).

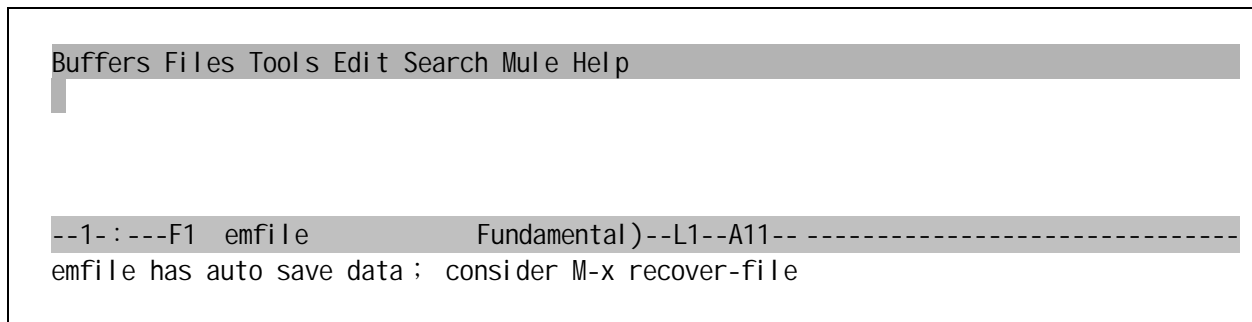



그림 5-1. emacs화면

방식행에는 파일이름(emfile)과 유표위치(L1)와 같은 쓸모 있는 많은 정보들이 들어 있다. F1의 왼쪽에 있는 3개의 이음표(---)들중 2개는 본문을 입력하는 순간에 변경될것이다.

최종행(소형완충기)은 사용자들이 emacs지령들을 입력하는데 리용된다. emacs는 체계통보문들을 현시하는데도 최종행을 리용한다. 아래의 그림에서는 편집기의 독특한 기능인 파일의 자동보관을 표현하는 통보문을 보여 주고 있다.

주해

emacs판본이 우에 차림표를 가지고 있는 경우 X Window체계의 xtem창문(12.9)에서 이 편집기를 실행시킬 때 그 차림표를 사용할수 있다. 많은 emacs지령들이 여기로부터 호출될수 있다.

5.1.1 조종건과 메타건

vi와 달리 그리고 대부분의 문서편집기들처럼 emacs는 방식 없는 편집기(mode-less editor)이다. 지령방식(vi용어)지령들을 입력하기 위하여 개별적인 건(vi에서 [Esc])을 누를 필요는 없다. 본문삽입을 시작하기 위하여 어떤 건(vi에서 i처럼)을 눌러야 하는것도 아니다. 어떤 점에서 볼 때 emacs는 항상 입력방식(vi용어)이며 어느 때든 본문입력을 시작할수 있다.

emacs는 항상 입력방식에 있으므로 본문에 작용할 지령들을 실행시키기 위해서는 조종건들을 사용하여야 한다. emacs의 모든 기능들은 조종건들의 수많은 조합들에 의하여 수행된다. emacs는 또한 메타건들도 사용하는데 emacs문서를 찾아 보면 아래와 같이 서술된 건누름렬을 발견하게 될것이다.

```
C-e—[Ctrl-e]
C-x C-b—[Ctrl-x][Ctrl-b]
```

C-x b—[Ctrl-x]b는 [Ctrl-x][Ctrl-b]와 다르다
M-e—[Meta-e]

이러한 건누름렬이 많으며(vi보다 훨씬 더 많다.) 그 대부분은 기억하기도 힘들고 비직관적이다. 또 한 건이 눌리워 져 있게 되는가 어떤가에 대한 판단에서도 혼동되기 쉽다.

5.1.2 건들의 사용법

편집기지령들을 입력하기 위해서는 3개의 특수건 [Ctrl], [Meta], [Esc]건들이 필요하다. 이 건들은 일반문자(인쇄가능한 문자)와 함께 사용되며 추가적으로 또 다른 건과 조합될수도 있다. PC건반에서는 [Mata]건을 보지 못했겠지만 그 건을 대신할수 있는 가능성에 대해서 후에 론의하기로 하자.

조작순서 C-e를 고찰하여 보자. 이것은 [Ctrl-e]일뿐이며 고심할 필요가 없다. C-x C-b는 사실상 [Ctrl-x][Ctrl-b]이며 다음과 같이 리용하여야 한다.

1. [Ctrl]건을 누른채로 유지한다.
2. x와 b를 차례로 누른다.
3. [ctrl]과 b를 둘 다 놓는다.

C-x C-b는 [Ctrl-x]b와 명백히 다르다. [Ctrl-x][Ctrl-b]지령은 완충기들을 렬거하며 [Ctrl-x]b는 [Ctrl-x][Ctrl-b]에 대한 스위치로서 사용된다. [Ctrl-x]b를 사용하려면 건을 다른 순서로 눌러야 한다.

1. 보통방법으로 [Ctrl-x]를 입력한다.
2. 두 건을 놓는다.
3. b만 누른다.

주해

일반화된 조작순서 [Ctrl-x1][Ctrl-x2]는 [Ctrl-x1]x2와 다르며 여기서 x1과 x2는 두개의 일반문자들이다. 첫 경우에는 x2를 누를 때까지 [Ctrl]건을 누르고 있어야 하지만 두번째 경우에는 x2를 누를 때에는 [Ctrl]건을 누르지 말아야 한다.

emacs에서 조작순서 M-e는 [Meta]건을 가진 건반들에서 [Meta-e]를 표현하는것으로 리해한다. 대부분의 체계들의 건반에는 이 건이 없지만 [Meta]건대신 사용할수 있는 건들을 제공한다. 만일 싼워크스테이션에서 [Alt]건이 동작하지 않는다면 다이아몬드기호를 가진 건을 사용하여야 할것이다.

PC에서 emacs를 사용하고 있을 때에는 [Alt]건을 쓸수 있다. 많은 경우 [Alt]건이 동작할것이며 M-e가 보이면 그대신 [Alt-e]를 사용할수 있다. [Alt]건이 동작하지 않는 경우에는 또 다른 방법으로서 [Esc]건을 리용할수 있다. 그러나 여기서 주의하여야 할것은 [Esc]건을 눌렀다 놓은 다음 e를 누르는것이다. 이 장에서는 [Alt]를 사용하여 [Meta]를 표현하였지만 매 사용자들은 실제적으로 자기들의 체계상에서 동작하는 건을 사용하여야 한다.

[Ctrl]과 [Meta]건을 둘 다 사용하는 조작순서들이 있다. M-5 C-f를 실행하려면 아래의 두 방법가운데서 어느 한가지 방법을 리용해야 할것이다.

[Alt-5]를 누르고 그 건을 놓은 다음 [Ctrl-f]를 누른다.

[Esc]를 눌렀다 놓은 다음 5를 누르고 그 건을 놓은 다음 [Ctrl-f]를 누른다.

[Alt]건이 체계에서 동작한다면 [Esc]건보다 사용하기 쉽다는것을 체험할수 있다. 그러한 경우에는 [Esc]건을 좀처럼 사용하려 하지 않을것이다. 비록 이 장에서는 [Ctrl], [Esc], [Alt]를 사용하고 있다고 할지라도 사용자들은 emacs형식(C-, ESC, M-)으로 사용하게 될것이다. 왜냐하면 그것들이 emacs를

서술하는 많은 본문들과 문서들에서 그렇게 표현되기때문이다.

주해

[Ctrl-x]를 누르고 다음건을 누를 때까지 1초이상 기다리면 소형완충기에 C-x-가 나타난다. 이것은 지령이 아직 완성되지 않았다는것을 의미한다.

5.1.3 직접적인 지령입력([Alt-x])

유효한 건들의 렬을 누르면 emacs는 그 건과 관련되는 지령을 실행한다. 실례로 [Ctrl-n]을 누르면 emacs는 이 건에 속박되여 있는 next-line지령을 실행한다. 물론 [Ctrl-n]과 next-line지령사이는 **건맺기**(key binding)가 존재하기때문에 이 긴 순서렬을 입력하는것보다 오히려 [Ctrl-n]을 사용하는 편이 낫다.

emacs는 매우 강력한 편집기이며 수천개의 이런 지령들을 가지고 있다. 그중 일부는 nonincremental-repeat-search-forward와 같이 매우 긴 단어들이다. 명백히 이런 2~3개의 지령들만은 대응되는 건들을 가진다.

모든 emacs지령들은 먼저 [Alt-x]를 누른 다음 지령본문을 입력하고 [Enter]를 누르는 형식으로 입력된다. next-line지령을 리용하는 방법을 아래에 준다.

```
[Alt-x]next-line[Enter]          실제상 M-x
```

[Alt-x]를 누를 때 소형완충기에는 문자렬 M-x가 나타난다. 문자렬 next-line을 입력한 다음 [Enter]를 누른다. 유표는 한행만큼 아래로 내려 간다. [Alt]건이 동작하지 않는다면 [Esc]xnext-line[Enter]를 리용하여야 한다. x를 누르기전에 [Esc]를 놓아야 한다.

emacs지령을 실행할 때마다 긴 단어들을 입력하여야 하는가? 완전본문을 입력할 필요는 전혀 없다. emacs는 완성하기기능을 가지고 있으며 그 기능이 사용자작업의 일부를 수행할것이다.

참고

지령을 직접 입력할 때 리용할수 있는 훌륭한 리력기능이 있다. emacs는 [Alt-x]나 [Esc]x로 명백하게 표현된 모든 지령들을 저장하며 작업의 전과정을 기억한다. 먼저 [Ctrl-x]를 누른 다음 Esc를 두번 누르는것으로 임의의 지령을 재호출할수 있다. 그다음 소형완충기에 나타난 마지막지령을 볼수 있다. Up건을 사용하여 종전의 지령들을 재호출하고 편집하여 실행한다. vim은 최종행방식에서 작업하는것과 류사한 기능을 가지고 있다.

5.1.4 건입력렬의 취소([Ctrl-g])

emacs는 대화식의 편집기이다. emacs는 자주 소형완충기(마지막행)에서 입력을 얻는다. emacs지령의 본문이나 탐색에 리용될 파일이름, 표현이 요구될수도 있다. 처음에는 실수를 하거나 틀린 건을 누를수도 있다. 사용자는 emacs가 무엇인가를 기다리고 있을 때에는 현재의 지령을 취소할수 있어야 한다. 이것을 [Ctrl-g]로 수행한다.

실례로 파일을 여는 emacs지령은 [Ctrl-x][Ctrl-f]이다. 다른 파일을 편집하려 할 때에는 [Ctrl-x]대신에 [Alt-x]를 누를수 있다. emacs는 그때 파일이름보다도 지령본문을 요구한다.

```
M-x              모든 emacs지령들은 여기서 입력된다
```

이것은 사용자가 원하는것이 아니므로 이 프롬프트를 제거하고 emacs가 보통방식으로 돌아 가게 하여야 한다. 이런 상태에 있을 때마다 [Ctrl-g]를 누른다. 바라는대로 되지 않으면 [Ctrl-g]를 두번 사용한다.

5.1.5 .emacs파일

많은 UNIX지령들은 기동할 때 구성파일을 읽는다. vi가 .exrc를 사용하는것처럼 emacs도 .emacs 파일을 사용한다. .emacs의 항목들은 LISP(원래 emacs를 서술하기 위하여 사용된 언어)로 서술된다. 이 항목들은 쉽게 읽을수 없다.

사용자의 가입등록부에는 대체로 .emacs가 포함되여 있다. 이 파일에는 또한 이 장에서 취급한 기정값들을 변경시키는 항목들도 포함될수 있다. 이것은 일부 지령들이 다르게 작업하든가 또는 전혀 작업하지 않도록 한다. 그런 일이 일어 날 가능성을 배제하자면 mv지령을 리용하여 파일이름을 다른것으로 바꾸어야 한다.

```
$ mv .emacs .emacs.bak          이름 .emacs를 .emacs.bak로 바꾼다
$ _
```

선택적으로 -q선택항목을 가지고 emacs를 호출할수 있다. 이 선택항목은 기동할 때 .emacs파일을 무시한다.

참고

사용자가 .emacs파일의 이름을 고칠수 있다면 -q선택항목은 요구되지 않는다. 그러나 .emacs를 사용하는 방법을 배우기전까지는 emacs를 -q선택항목과 함께 호출하도록 하자.

5.2 emacs에서의 탈퇴

우리는 파일이 아니라 **완충기**(그 파일의 복사본)를 가지고 편집을 진행한다는것을 잘 알고 있다. 완충기는 그 파일과 분리되여 존재한다. 완충기에 대한 변경은 보관조작에 의하여 디스크파일에 씌여 진다.

emacs에서 작업을 시작하기전에 편집기에서 탈퇴하는 방법을 배워야 한다. 아래에 그 두가지 방법을 보여 준다.

- 파일에 완충기변경내용들을 쓰고 탈퇴한다.
- 완충기에 대한 모든 변경내용들을 포기하고 탈퇴한다.

emacs는 하나의 지령으로 이 두가지 경우를 조종한다. [Ctrl]건을 누른 상태에서 x와 c를 차례로 누른다.

```
[Ctrl-x][Ctrl-c]          C-x C-c
$ _
```

편집내용을 마지막으로 보관한후에 완충기에 아무런 변경도 가해 지지 않았다면 즉시 쉘프롬프트로 귀환한다. 그러나 변경된것이 있다면 emacs는 다음과 같은 통보문을 발생시킨다.

```
Save file /home/romeo/project5/emfile? (y, n, !, ., q, C-r or C-h)
```

이제는 변경내용들을 보관하겠는가를 결정하여야 한다. 여기에 6개의 선택항목이 있지만 그것들을 모두 알 필요는 없다. y를 누르면 파일을 보관하고 편집기에서 탈퇴한다.

때때로 사용자는 이 질문에 대답하지 않고 될수록 빨리 편집내용을 보관하고 탈퇴하려고 할수도 있다. 그를 위한 특정한 조작순서는 다음과 같다.

```
[Ctrl-u][Ctrl-x][Ctrl-c]          질문을 요구하지 않는다
```

이 지령은 세개의 조종건들을 사용한다. 따라서 [Ctrl]건을 누른 상태에서 u, x, c를 차례로 누른다.

편집과정의 무시

얼마간의 본문을 입력하였지만 후에 그것을 보관하지 않겠다고 결심할 때가 종종 있다. 그러한 경우에는 그 작업을 무시(abort)해야 한다. 수정된 완충기에서 [Ctrl-x][Ctrl-c]를 사용하면 우에서 보여 준 것과 같은 질문이 나타난다. 이때 n을 누르면 emacs의 보호기구가 다시한번 확인한다.

```
Modified buffers exist; exit anyway? (yes or no)
```

지금 emacs는 완충기가 마지막으로 보관된후에 수정되였다는것을 알려 준다. 이때 yes 또는 no(y나 n이 아니라)를 입력하여야 한다. yes는 편집과정을 중지하고 쉘로 귀환하며 no는 편집작업을 계속한다.

5.3 본문의 삽입과 치환

emacs를 호출하면 유표는 첫행의 첫 문자에 배치된다. 그림 5-2에 보여 준것과 같이 몇행의 본문을 삽입하여 보자. 다섯개의 행을 입력한후에 방식행에 나타나는 현재유표가 놓여 있는 행번호(L5)에 주의를 돌리자. 현재 사용자는 파일의 웃부분(Top)에 위치하고 있으며 왼쪽의 두 별표는 완충기의 내용이 수정되였다는것을 나타낸다. 여기서 다른 기호들은 무시하기로 하자.

```
Buffers Files Tools Edit Search Mule Help
This is the emacs editor
When you start it without specifying a filename,
the system will show you a screenful of text
press the spacebar to clear the window and then start editing
If you ger stuck,then use C-x C-c to quit.
 …blank lines deleted…
--1-:**-F1 emfile              (Fundamental)--L5--Top----------------------------
```

그림 5-2. 본문입력

왼쪽의 본문을 지우기 위하여서는 [Ctrl-h]가 아니라 [BackSpace]건을 사용할수 있다. [Ctrl-h]는 emacs가 도움말기능을 호출하기 위하여 사용하는 건이다. emacs는 상당히 풍부한 도움말기능을 가지고 있으며 [Ctrl-h]를 리용하여 그것을 동작시킨다. 유표아래문자를 지우기 위하여서는 [Ctrl-d]나 [Delete]를 사용할수도 있다.

주의

빈 파일인 경우 방향건을 리용하여 수직방향으로는 이동할수 있다. 아래로 내려 가기 위하여 DOWN건을 누를 때마다 emacs는 빈 행을 하나씩 추가한다. 이때 주의하지 않으면 [Enter]건을 누르지 않아도 빈 행들이 불필요하게 추가된다.

5.3.1 본문의 치환(overwrite-mode)

기정적으로 emacs는 **삽입방식**(insert mode)에서 작업하며 이것은 본문이 입력될 때마다 현존본문이 오른쪽으로 밀려 난다는것을 의미한다. emacs는 **겹쳐쓰기방식**(overwrite mode)으로도 동작하는데 이 방식은 유표가 지나가는 모든 문자들을 치환한다. 이 방식으로 절환하는데는 다음과 같은 두가지 방법이

있다.

- 반전스위치(toggle switch)로서 동작하는 [Insert]건을 누르는 방법. 한번 누르면 겹쳐쓰기방식으로 절환되며 다시 한번 누르면 삽입방식으로 돌아 온다.
- emacs overwrite-mode지령을 리용하는 방법. [Alt-x]를 누르면 emacs는 소형완충기에 M-x를 표시하고 입력을 기다린다. 이때 overwrite-mode를 입력하고 [Enter]건을 누른다.

```
M-x overwrite-mode[Enter]
```

이 지령도 반전효과를 가지며 이 지령을 다시 사용하여 삽입방식으로 돌아 갈수 있다. 이것은 많은 건조작을 요구하지만 지령완성하기기능을 리용하면 그 본문의 일부만을 입력한 다음 [Tab]건을 눌러 그 지령을 완성할수 있다. 그에 대하여서는 곧 취급하게 될것이다.

주해

겹쳐쓰기방식에서는 방식행에 Ovwrt라는 단어가 나타난다. 삽입방식에 들어 가려면 [Insert]건을 누르거나 [Alt-x]와 함께 overwrite-mode지령을 입력한다.

5.3.2 조종문자들의 입력([Ctrl-q])

emacs는 모든 경우에 조종건들을 사용한다. 그것은 조종건을 누르는것만으로 emacs지령들을 실행시킬수 있기때문이다. 조종문자들을 입력하기 위해서는 먼저 [Ctrl-q]를 눌러야 한다. [Ctrl-m]을 입력하기 위하여서는 다음과 같은 조작순서를 사용하여야 한다.

```
[Ctrl-q][Ctrl-m]          ^M이 나타난다
```

이것은 화면에 ^M으로 나타난다(그림 5-3). vi부분에서 서술한바와 같이 이것은 2개의 개별적인 문자처럼 보여도 실제로는 한개의 문자이다. [Esc]건도 같은 방법으로 입력된다([Ctrl-q][Esc]).

```
A DOS file has a Ctrl-m at the end of each line
  [Ctrl-q][Ctrl-m]
A DOS file has a Ctrl-m at the end of each line^M
```

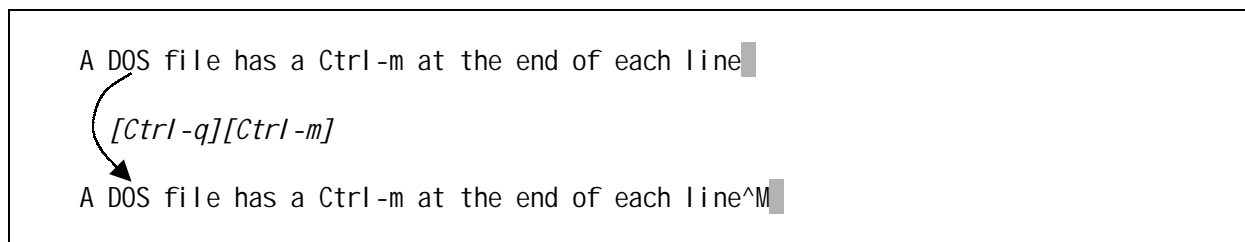



그림 5-3. 조종문자삽입

5.4 본문의 보관

비록 emacs가 자동적으로 파일을 보관(다른 파일이름으로)한다 해도 사용자가 자기의 작업내용을 규칙적으로 보관하여야 한다. 이를 위한 지령은 다음과 같다.

```
[Ctrl-x][Ctrl-s]          같은 파일에 보관한다
```

이 지령은 완충기를 emacs를 호출할 때 사용하였던 그 파일에 보관한다. 때로는 다른 이름을 제공하려고 할수 있는데 그런 경우에는 다음과 같은 지령을 사용하여야 한다.

```
[Ctrl-x][Ctrl-w]          다른 파일에 보관한다
```

체계는 현재등록부를 보여 주며 파일이름을 요구한다. 이때 다른 파일이름(실례로 emfile.txt와 같

은)을 입력하고 [Enter]를 누르자.

```
Write file: ~/project5/emfile.txt
```

그러면 방식행에 새로운 파일이름이 나타나며 현재완충기도 그 이름과 련결된다. 다른 파일에 쓸 때 emacs의 동작은 vi와 명백한 대조를 이룬다. 즉 vi는 새로운 파일을 현재파일로 보지 않는다. 이 방식에서 기본적인 보관 및 탈퇴지령들을 표 5-1에 보여 준다.

표 5-1. emacs의 보관, 탈퇴지령들

지 령	기 능
[Ctrl-x][Ctrl-s]	파일을 보관하고 편집방식을 유지한다
[Ctrl-x][Ctrl-w]	다른 파일로 보관한다(MS Windows에서의 Save As ...와 같다)
[Ctrl-x][Ctrl-c]	파일이 변경되지 않았을 때 또는 그다음의 프롬프트에 n과 yes가 입력되였을 때(파일이 수정되였다면) 편집방식에서 탈퇴한다
[Ctrl-u][Ctrl-x][Ctrl-c]	프롬프트없이 보관하고 탈퇴한다
[Ctrl-x][Ctrl-z]	현재작업을 중지하고 UNIX쉘에로 림시탈퇴한다(fg 또는 exit를 사용하여 emacs에로 돌아 간다)
[Ctrl-z]	우와 같다
[Alt-x]shell	현재의 창문에서 UNIX쉘에로 림시탈퇴한다(emacs에로 돌아 가려면 [Ctrl-x]b[Enter]를 사용한다)

참고

때때로 완충기내용을 현재파일에 보관할수 없을수도 있는데 이것은 파일이 쓰기보호되여 있기때문이다. 그러한 경우에는 [Ctrl-x][Ctrl-w]를 사용하여 다른 파일에 보관하여야 한다.

파괴로부터의 회복

vi와 달리 emacs는 300개의 건이 입력될 때마다(또는 사용자가 아무것도 하지 않는 경우에는 30초마다) 완충기의 내용을 자동적으로 보관하는 **자동보관**(autosave)기능을 가지고 있다. 결국 체계가 파괴되는 경우 기껏해서 300개의 건작업내용만을 잃어 버릴수 있다. 그러나 emacs는 같은 파일이 아니라 원래의 파일이름의 량끝에 #를 붙인 이름에 대하여 자동보관을 진행한다. 실례로 파일 emfile은 현재등록부의 #emfile#에 자동보관된다.

전원이 갑자기 꺼지거나 체계가 파괴되는 경우에 사용자는 완충기의 최종상태를 보관하지 못할수도 있다. 이때에는 recover-file지령을 사용하여 현재완충기의 내용을 마지막으로 자동보관된 파일로 교체할수 있다.

```
[Alt-x]recover-file
```

파일이름(량쪽에 #가 있어야 한다.)을 입력하면 emacs는 두 파일의 크기와 마지막으로 변경된 시간을 보여 준다. 또한 완전히 단어화된 대답을 요구한다. 그 프롬프트를 보여 주는 화면을 그림 5-4에 보여 준다.

```
-rw-r--r--   1 romeo  dialout    2658  Sep 19 09:14 /home/romeo/p5/#note1#
-rw-r--r--   1 romeo  dialout    2677  Sep 19 09:06 /home/romeo/p5/note1
...
--1-:%%-F1   *Directory      (Help View)--L1--All-------------------------------------
Recover auto save file /home/remeo/p5/#note1#? (yes or no) yes
```

그림 5-4. recover-file로 파일되살리기

많은 변경을 진행한 다음 그 변경들중 일부가 잘못되였다는것을 발견했을 때 자동보관기능을 유용하게 리용할수 있다. 그림에서 사용자는 마지막으로 보관한것보다 8분이나 더 새로운 판본을 가지고 있으며 그것을 리용하려고 할수도 있다. 또한 emacs는 기동시에 자동보관판본이 원래판본보다 더 최근의것일 때에는 사용자에게 그에 대하여 알려 준다.

5.5 수자인수

편집기능의 사용을 시작하기전에 emacs가 지령에 수를 붙이는 기능도 지원한다는것을 알아야 한다. 이 수를 **수자인수**(digit argument)라고 하며 해당 지령을 여러번 수행하도록 하는 기능을 수행한다. 여기서 수자의 앞에는 [Meta]건이 놓인다.

[Ctrl-d]는 한 문자를 지운다. 그러므로 5개의 문자를 지우기 위하여서는 다음과 같은 지령을 주어야 한다.

```
[Meta-5][Ctrl-d]        5는 수자인수이다
```

이 지령은 다음과 같은 두가지 방법으로 수행된다.

```
[Alt-5][Ctrl-d]         [Alt]를 누른 상태에서 5를 입력한다
[Esc]5[Ctrl-d]          5를 입력하기전에 [Esc]를 놓는다
```

그림 5-5에 수자인수를 써서 문자지우기지령을 사용한 경우와 수자인수를 쓰지 않고 문자지우기지령을 사용한 경우의 실례를 보여 준다.

```
Try X-Window emacs if you have trouble learning emacs

        [Ctrl-d]

Try XWindow emacs if you have trouble learning emacs

        [Alt-6][Ctrl-d]

Try X emacs if you have trouble learning emacs
```

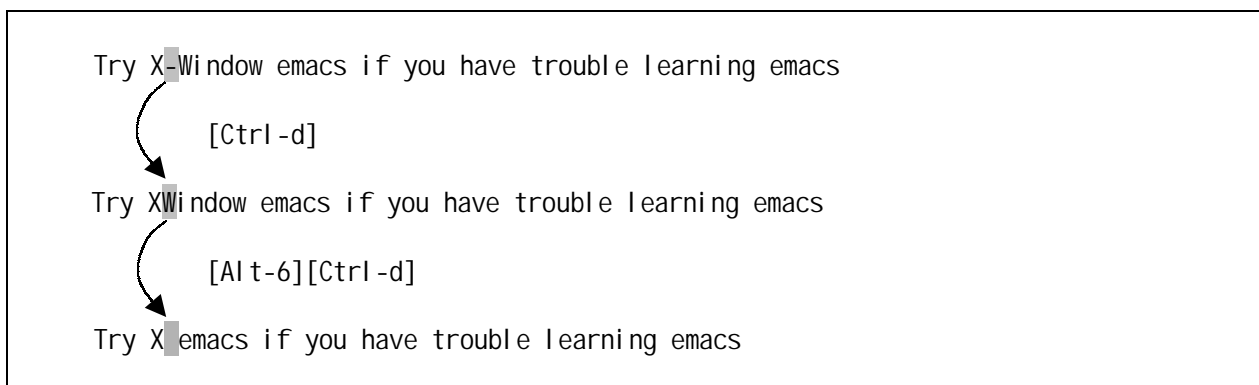



그림 5-5. [Ctrl-d]로 문자지우기

만능인수 - [Ctrl-u]

수자인수를 입력하는 다른 한가지 방법으로서 [Ctrl-u]뒤에 임의의 수를 리용하는 방법이 있다. 다음의 지령렬도 5개의 문자를 지운다.

```
[Ctrl-u]5[Ctrl-d]             [Alt-5][Ctrl-d]와 같다
```

이 지령은 필요 없는 작업인것처럼 보이지만 [Ctrl-u]는 특수한 건조작이며 이것을 **만능인수**(universal argument)라고 부른다. 수자인수와 꼭 함께 사용할 필요는 없으며 요구되는 회수만큼 반복할수 있다. [Ctrl-u][Ctrl-d]를 사용하면(수자인수가 없이) 지우기지령이 4번 실행된다. 추가되는 매 [Ctrl-u]는 반복회수를 4배로 증가시킨다. 다음의 두 조작순서는 각각 16개 문자와 64개 문자를 지운다.

```
[Ctrl-u][Ctrl-u][Ctrl-d]      16개 문자를 지운다
```

[Ctrl-u][Ctrl-u][Ctrl-u][Ctrl-d]　　　　64개 문자를 지운다

3자리수자나 4자리수자를 입력하는것보다 같은 건을 몇번 누르는것이 더 쉽기때문에 때로는 이 방법이 오히려 더 나을수 있다. [Ctrl-u]는 작업의 일부분을 수행하는데도 사용할수 있다. 66개 문자를 지우기 위하여서는 [Ctrl-d]와 함께 [Ctrl-u]를 세번 사용하여 64개 문자를 지운 다음 [Ctrl-d]를 두번 사용할수 있다. 만능인수의 사용법을 그림 5-6에 보여 준다.

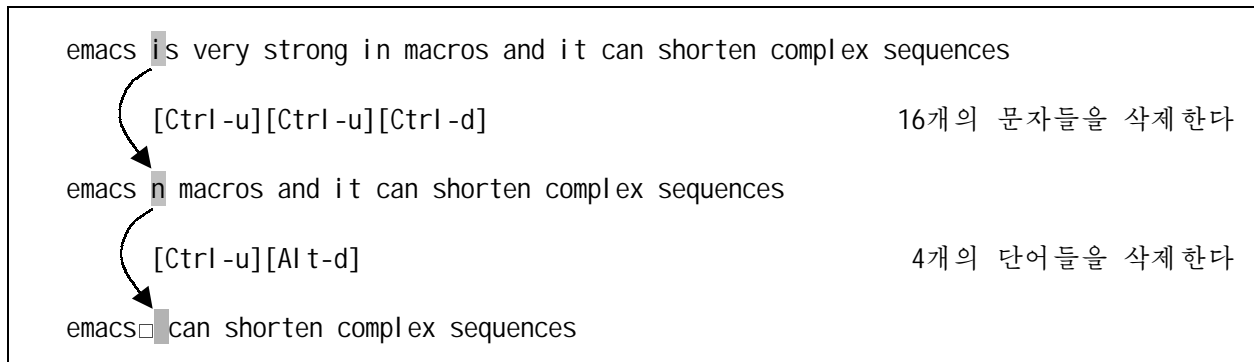



그림 5-6. [Ctrl-n]을 만능인수로서 사용

5.6 항행

emacs는 조종건들을 사용하여 4개의 방향으로 이동한다. 이 건들은 매우 직관적이다. 즉 [Ctrl-b]는 유표를 왼쪽으로(back), [Ctrl-f]는 오른쪽으로(forward), [Ctrl-p]는 유표를 우로(previous line), [Ctrl-n]은 아래로(next line) 이동시킨다.

수자인수는 여기에도 적용된다. 즉 [Alt-4][Ctrl-b]는 유표를 4자리뒤로 이동시키며 [Alt-200][Ctrl-n]은 200행아래로 내려 간다. [Ctrl-n]과 다른 3개의 조종건들사이에는 뚜렷한 하나의 차이점이 있다. 완충기의 끝에 이르면 [Ctrl-n]은 Down건을 누를 때(5.3의 주의)처럼 빈 행을 추가한다. 기본적인 항행(navigation)지령들을 그림 5-7에 보여 준다.

주의

완충기끝에 이르면 필요없이 [Ctrl-n]을 누르지 않도록 주의해야 한다. [Ctrl-n]은 완충기에 행들을 계속 추가하기만 한다. 이 방법으로 많은 빈 행들이 생긴다.

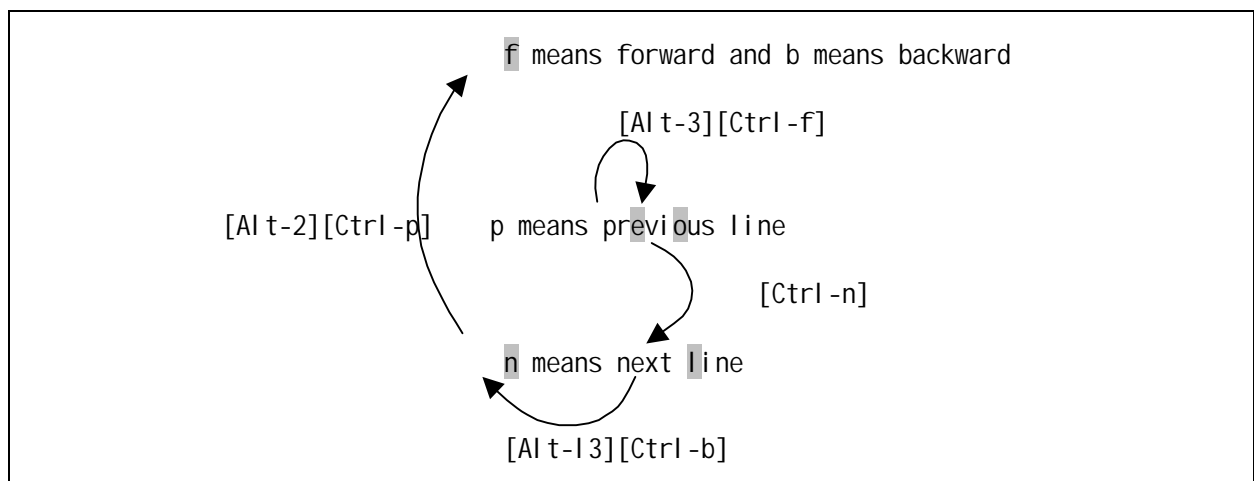



그림 5-7. 수평수직이동

5.6.1 페지화와 화면흘리기([Ctrl-v]와 [Alt-v])

화면을 우아래로 흘려 보내기(scrolling) 위한 건은 v이다. v는 우로 흘리는가 아래로 흘리는가에 따라 [Ctrl]이나 [Alt]건과 함께 사용된다. 우로 흘리려면 [Ctrl-v]를 사용하고 아래로 흘리려면 [Alt-v]를 사용한다.

여기에도 수자인수가 적용되며 한번에 여러개의 화면을 흘려 보낼수 있다. 화면이 헝클어 지면 [Ctrl-l]을 사용하여 화면을 다시 그려야 할것이다. vi에서도 같은 건조합을 사용한다. 그러나 한가지 차이점이 있다. emacs에서 [Ctrl-l]은 현재행을 화면의 중심으로 옮긴다. 이것은 유표를 현재행밖으로 이동함이 없이 본문을 더 잘 볼수 있게 하여 준다. 유표의 수평수직이동과 페지화지령들을 표 5-2에 보여 준다.

표 5-2. emacs에서 수평수직이동

지 령	기 능
[Ctrl-b]	유표를 우로 이동시킨다
[Ctrl-f]	유표를 오른쪽으로 이동시킨다
[Ctrl-p]	유표를 우로 이동시킨다
[Ctrl-n]	유표를 아래로 이동시킨다(완충기의 끝에서 실행될 때에는 빈 행들을 추가한다)
[Ctrl-v]	전화면 앞으로 흘린다
[Alt-v]	전화면 뒤로 흘린다
[Ctrl-l]	화면을 지우고 유표를 화면의 중심에 배치한다

5.6.2 행의 시작과 끝([Ctrl-a]와 [Ctrl-e])

사용자는 행의 시작이나 끝으로 매우 빨리 이동할수 있다. 이것은 [Ctrl-a]나 [Ctrl-e]건으로 조종한다. 행의 시작으로 이동하려면 [Ctrl-a](vi에서 0과 같다.)를 사용하며 행의 끝으로 이동하려면 [ctrl-e](vi에서 $와 같다.)를 사용한다.

여기서 수자인수들이 동작하는가 하는것은 자체로 검사하기로 한다. 이 두개의 지령이 단어단위를 리용하는 지령([Alt-f]와 [Alt-b])들과 함께 사용되는 실례를 그림 5-8에 보여 준다.

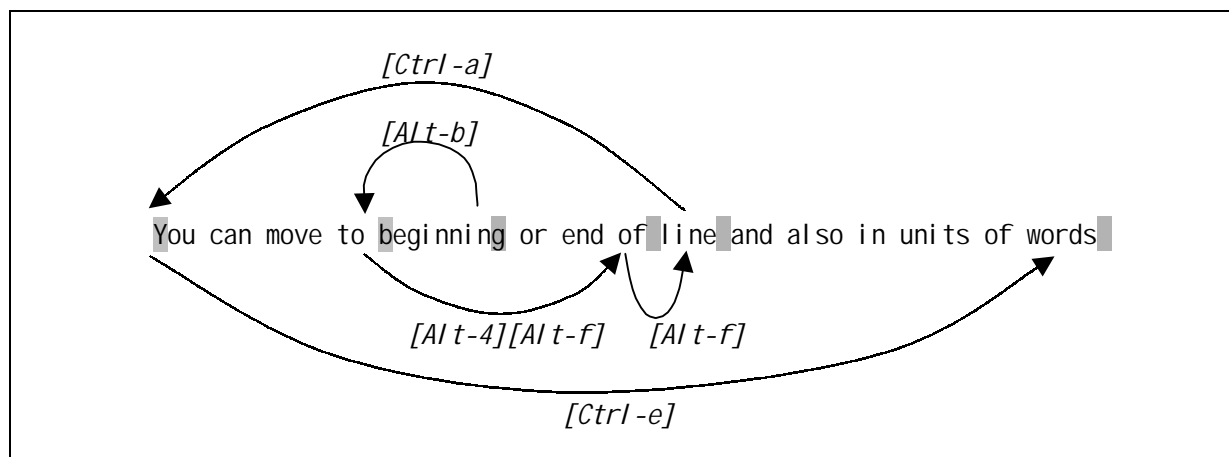



그림 5-8. 행안에서의 항행

5.6.3 단어단위의 항행([Alt-f]와 [Alt-b])

emacs는 단어를 항행단위로 인식하기도 한다. 이때 하나이상의 단어들에 관하여 이동할수 있다. 단

어의 시작으로 이동하려면 [Alt-f](한 문자만큼 앞으로 이동하려면 [Ctrl-f])를 사용하며 한 단어만큼 뒤로 이동하려면 [Alt-b](한 문자만큼 뒤로 이동하려면 [Ctrl-b])를 사용한다.

수자인수는 행에서의 항행을 빠르게 해준다. 실례로 [Alt-5][Alt-f]는 유표를 5개 단어만큼 전진시키며 [Alt-3][Alt-b]는 3개 단어만큼 뒤로 이동시킨다.

5.6.4 행사이의 이동

emacs는 기정으로 방식행에 현재행번호를 보여 준다(앞에 L을 붙인다). 만일 체계가 행번호를 표시하지 않으면 Line-number-mode지령을 줄수 있다. 이 지령도 특별한 건입력렬에 결부되지 않은 또 하나의 emacs지령이다. 그러므로 이 지령을 사용하기전에 [Alt-x]를 입력해야 한다.

```
[Alt-x]line-number-mode[Enter]
```

그러면 방식행에 현재행번호와 함께 L이 나타난다. 이 지령도 반전스위치로서 그 지령을 다시 주면 행번호가 나타나지 않는다.

goto-line지령을 리용하여 특정한 행번호에로 이동할수 있다. 40행으로 유표를 이동하려면 다음의 지령을 사용한다.

```
[Alt-x]goto-line[Enter]
```

emacs는 이때 행번호를 입력할것을 요구한다.

```
Goto line: 40[Enter]            vi에서의 40G와 같다
```

건조작이 많은것 같이 보이지만 완성하기기능이 어떻게 본문입력을 단축하는가를 곧 알게 될것이다. 또한 다음의 지령을 입력하여 프롬프트없이 이행할수도 있다.

```
[Ctrl-u]40[Alt-x]goto-line[Enter]
```

이 지령은 건조작이 많으므로 그것을 간단화하려면 건반의 어떤 건에 대응시켜야 한다. 이 장의 뒤부분에서 goto-line지령에 대하여 다시 설명할것이다.

참고

만일 편집기를 파일과 함께 기동할 때 유표가 일정한 행번호(례를 들어 40행)에 위치하도록 하려면 행번호앞에 +를 붙여 emacs의 인수로 사용하여야 한다

```
emacs +40 Chap01
```

5.6.5 본문의 시작과 끝으로의 이동([Alt-<]와 [Alt->])

emacs는 파일의 시작이나 끝으로 이동하는 특수한 건입력렬을 지원한다. 행번호 1로 가려면 [Alt-<](vi에서 1G와 같다.)를 사용하며 파일의 끝으로 가려면 [Alt->](vi에서 G와 같다.)를 사용한다.

PC건반에서 기호 <와 >는 [Shift]건을 눌러 호출한다. 우에서 보여 준 두 지령을 사용하려면 실제적으로는 [Alt]와 함께 [Shift]를 사용하여야 한다.

```
[Alt][Shift]<          파일의 시작
[Alt][Shift]>          파일의 끝
```

지금까지 배운 항행지령들을 표 5-3에 보여 준다.

표 5-3. emacs의 항행지령들

지 령	기 능
[Ctrl-a]	행의 시작점으로 이동한다
[Ctrl-e]	행의 끝으로 이동한다
[Alt-b]	단어의 시작점으로 후진한다
[Alt-f]	한 단어만큼 전진한다
[Alt->]	파일의 끝으로 이동한다
[Alt-<]	파일의 시작으로 이동한다
[Alt-x]goto-line[Enter]40	40행으로 이행한다(프롬프트에 행번호를 입력한다)
[Ctrl-u]40[Alt-x]goto-line	40행으로 이행한다(프롬프트없이)
[Alt-x]line-number-mode	방식행에서의 행번호현시방식을 절환한다

주해

<, >, |, ^와 같은 일부 건들을 호출하기 위하여서는 [Shift]건을 사용하여야 한다. 그러나 [Shift]와 함께 또는 [Shift]를 사용함이 없이 쓰이는 건조작들이 일부 있다([Ctrl-a]와 [Ctrl-_]). 이 장에서는 가령 지령이 요구하더라도 일반적으로는 [Shift]를 지정하고 있지 않다. 자체로 결심하는것은 어렵지 않으며 [Shift]를 지정하지 않아 작업이 실패하는 경우에는 [Shift]를 그 입력렬에 추가하면 될것이다.

5.7 구역을 리용한 작업

emacs는 본문의 블로크를 **구역**(region)으로서 표시하도록 한다. 이렇게 하는데는 마우스가 필요 없으며 2~3개의 건조작을 사용하면 구역을 정의할수 있다. emacs는 구역에서 동작하는 삭제, 대소문자변경, 복사기능들을 지원한다. 정의에 의하여 구역에서 작업하는 지령은 그 구역에 의하여 정의된 전체 본문에 작용한다.

구역을 정의하려면 그 구역의 량끝을 표식하기만 하면 된다. 구역의 시작으로 될 위치에 유표를 배치하고 다음과 같은 건누름렬들중 하나를 누른다.

```
[Ctrl][Spacebar]        표식설정
[Ctrl-@]        -[Shift]를 사용하거나 또는 사용할수 없다
```

이 지령은 눈에 보이지 않는 **표식**(mark)을 설정한다. 이때 조종건들을 사용하여 유표를 그 단락의 끝으로 이동시키기로 하자. 이 처리는 본문을 강조하지도 않으며 표식을 보여 주지도 않지만 자동적으로 구역을 정의한다. 두 끝사이를 왔다갔다하려면 다음의 지령을 사용한다.

```
[Ctrl-x][Ctrl-x]        vi에서 ''와 같다
```

이때 유표가 앞뒤로 이동하는데 이것은 구역이 정의되였다는것을 보여 준다. 이제는 이 구역에서 임의의 지령들을 실행할수 있다.

emacs에서는 유표의 현재위치(구역의 끝에서)를 **점**(point)이라고 한다. 유표와 점사이에는 사실상 차이점이 없으며 단지 유표는 문자우에 항상 놓이고 점은 바로 그앞에 위치한다는것만이 다르다. 이 본문에서는 그러한 미묘한 차이는 무시하도록 할것이다.

참고

전체 완충기를 구역으로 표식하기 위하여 emacs는 지름건 [Ctrl-x]h를 제공한다. 표식과 점을 개별적으로 설정할 필요가 없는 경우에는 본문의 임의의 위치에서나 이 지령을 줄수 있다.

5.8 본문의 삭제, 이동, 복사

emacs는 문자나 단어, 행들을 삭제할수 있는 개별적인 지령들을 가지고 있다. 뿐만아니라 구역안에 정의된 본문도 지울수 있다. 한편 오직 구역내에서만 본문을 복사할수 있다. 먼저 구역을 요구하지 않는 지령들에 대하여 서술하고 그다음에 구역을 요구하는 지령들에 대하여 서술하도록 하자. 표 5-4에 앞으로 설명하게 될 편집지령들을 보여 준다.

표 5-4. emacs의 편집기능들

지 령	의 미
[Ctrl-d] 또는 [Delete]	한개의 문자를 지운다
[Alt-6][Ctrl-d]	6개의 문자를 지운다
[Alt-d]	한개의 단어를 지운다
[Ctrl-k][Ctrl-k]	현재행을 지운다
[Alt-6][Ctrl-k]	6개의 행을 지운다
[Ctrl-x][Ctrl-o]	뒤따르는 모든 빈 행들을 지운다
[Ctrl-@]	구역의 표식을 정의한다
[Ctrl][Spacebar]	우와 같다
[Ctrl-w]	구역안의 본문을 지운다
[Alt-w]	구역안의 본문을 복사한다
[Ctrl-y]	삭제 또는 복사된 본문을 유표의 오른쪽에 놓는다
[Ctrl-t]	문자를 앞의것으로 교환한다
[Ctrl-x][Ctrl-t]	현재행을 앞의 행과 교환한다
[Alt-^]	현재행을 앞의 행과 련결한다([Shift]가 필요)
[Alt-u]	단어를 대문자로 변환한다
[Alt-4][Alt-u]	4개의 단어를 대문자로 변환한다
[Alt-l]	단어를 소문자로 변환한다
[Alt-c]	단어의 첫 문자를 대문자로 변환한다
[Alt-5][Alt-c]	5개 단어의 첫 문자를 대문자로 변환한다
[Ctrl-x][Ctrl-u]	구역안의 전체 본문을 대문자로 변환한다
[Ctrl-x][Ctrl-l]	구역안의 전체 본문을 소문자로 변환한다
[Ctrl-x]u	마지막편집동작을 취소한다
[Ctrl-_]	마지막편집동작을 취소한다(조종건과 밑선)
[Ctrl- -]	마지막편집동작을 취소한다(조종건과 이음표)

5.8.1 단어와 행의 삭제([Alt-d]와 [Ctrl-k])

emacs에서 [Ctrl-d]는 한 문자를 삭제(delete)하며 단어를 지울 때는 그대신에 [Alt]건을 사용한다. 사용자들이 기억해 두어야 할 규칙은 [Alt]건이 더 큰 문자묶음(보통 단어들)에서 동작한다는것이다. 단어를 지우려면 다음의 지령을 사용한다.

```
[Alt-d]          vi에서 dw와 같다
```

단어묶음을 지우기 위하여서는 수자인수를 사용하여야 한다. 5개의 단어를 지우려면 다음과 같은 지령을 사용한다.

```
[Alt-5][Alt-d]          vi에서 5dw와 같다
```

이것은 5개의 단어를 **제거**(kill:삭제의 특수한 형식)한다. 여기서 **단어**(word)는 vi에서와 다르게 정

의되며 단순히 자모와 수자로 된 문자들의 렬이다(vi에서의 단어처럼 _문자를 포함하지 않는다).

행외에도 행의 시작이나 끝까지의 본문도 삭제할수 있으며 지어는 행자체를 지울수도 있다. 이 3가지 조작들에서 공통적으로 리용되는 건은 [Ctrl-k]이다. 현재유표위치부터 행의 끝까지의 본문을 삭제하려면 아래의 지령을 사용한다.

```
[Ctrl-k]         vi에서 d$와 같다
```

행의 시작까지의 본문을 삭제하려면 다음과 같이 [Alt-0]을 사용할 필요가 있다.

```
[Alt-0][Ctrl-k]         vi에서 d0과 같다
```

이상하게 여겨 질지 모르겠지만 행을 제거하는 간단한 지령은 없으며(vi에서의 dd와 달리) 행의 시작에서 [Ctrl-k]를 두번 사용하여야 한다. 이것은 먼저 행의 시작으로 이동하기 위하여 [Ctrl-a]를 사용한 다음 [Ctrl-k][Ctrl-k]를 사용하여야 한다는것을 의미한다.

첫 [Ctrl-k]는 행의 마지막에 나타난 문자까지 본문을 제거한다. 두번째 [Ctrl-k]는 첫 호출후에 남아 있는 행바꾸기(newline)문자를 제거한다.

이로부터 3개의 행(6개가 아니라)을 지우려면 [Alt-6][Ctrl-k]를 사용하여야 한다고 생각할수 있다. 얼핏 보기에는 옳은것 같지만 그렇치 않다. 6개의 행을 지운다고 하여 [Ctrl-k]가 더 요구되는것이 아니다.

```
[Alt-6][Ctrl-k]         vi에서의 6dd처럼 6개의 행을 지운다
```

emacs는 빈 행들의 묶음을 제거(removing)하는 특수한 기능을 지원한다. 이때 사용되는 지령은 다음과 같다.

```
[Ctrl-x][Ctrl-o]         빈 행들을 지운다
```

임의의 빈 행에서 이 지령을 실행하면 그 행을 제외한 모든 린접한 빈 행들이 제거된다. 빈 행의 바로 앞에 있는 비지 않은 행에서 이 지령을 실행하면 모든 빈 행들이 지워 진다. 행지우기를 그림 5-9에 보여 준다.

```
emacs is more than an editor
You can send mail from the editor
You can perform the directory functions
With enhancements you can use telnet and ftp
And the World Wide Web too

    [Ctrl-k][Ctrl-k]

You can send mail from the editor
You can perform the directory functions
With enhancements you can use telnet and ftp
And the World Wide Web too

    [Alt-3][Ctrl-k]

And the World Wide Web too
```

그림 5-9. 행지우기

5.8.2 제거고리

《삭제하다(delete)》와 《제거하다(kill)》라는 단어들을 서로 같은것으로 자유롭게 사용하는데 대하여 생각해 본적이 있는가? 정확히 말하면 그렇치 않다. emacs에서는 이 두 단어들사이에 명백한 차이를 가진다. [Ctrl-k]는 지워 진 본문을 **제거고리**(kill ring)라고 하는 저장령역에 보내기때문에 제거조작이라고 한다. 기정적으로 이 고리는 마지막 30개의 삭제(실제로는 제거)를 저장하며 이 고리에 보내여 진 본문은 후에 회복될수 있다.

한 문자나 공백을 지우는 지령들은 삭제범주에 해당된다. 이 지령들로 지운 본문은 제거고리에 보관되지 않는다. 한편 하나이상의 단어나 행을 지울 때에는 지워 진 본문이 제거고리에 보관된다. [Ctrl-d]를 제외한 모든 삭제조작들은 사실상 제거조작이다. 앞으로 우리는 제거조작의 경우에도 《삭제》라는 말을 사용할것이며 따라서 이 단어를 엄밀한 의미로 해석하기 위하여 주의할 필요는 없다.

제거고리는 제거조작에 의해 취해 진 본문만을 접수하는것은 아니며 복사된 본문도 제거고리에 보낼수 있다. 이에 대해서는 본문복사에 대하여 배울 때 취급할것이다.

주해

emacs는 련속삭제묶음을 하나의 묶음으로서 제거고리에 저장한다(다음번 비삭제조작이 수행될 때까지). [Alt-4][Alt-d]로 4개의 단어를 지우고 [Alt-8][Alt-k]로 8개의 행을 지운후에 [Ctrl-y](본문의 이동이나 복사을 위하여 사용하는 건)를 사용하면 모두 회복할수 있다. vi에서 p는 마지막으로 삭제된 8개의 행만을 되살린다.

여러개의 삭제된 단락들을 하나의 본문블로크로서 회복할수 있도록 하기 위하여서는 모든 삭제지령들이 차례로 실행되였는가(그 지령들사이에 하나의 비삭제지령도 없이)를 확인하여야 한다. 두 지우기조작사이에 유표를 조금 이동하였다면 그 련쇄고리가 깨여 진것으로 된다. 그렇게 되면 마지막비삭제조작후에 지워 진 본문들만 회복될것이다. 그러나 사용자는 제거고리를 하나하나 지나면서 그것들을 회복할수도 있다.

5.8.3 구역안에서 본문의 삭제([Ctrl-w])

본문의 임의의 단락들을 지우려면 앞에서 설명한것처럼 구역을 만들어야 한다. 표식과 점이 제대로 설정되였는가를 확인하고 [Ctrl-x][Ctrl-x]를 몇번 사용한 다음 [Ctrl-w]를 사용한다. 그러면 구역안의 본문이 삭제된다. 이때 구역의 삭제로 하여 생긴 공간은 점뒤의 본문으로 채워 진다. 구역안에서의 본문지우기를 그림 5-10에 보여 준다.

참고

전체 완충기의 내용을 지우려면 [Ctrl-x]h를 사용하여 구역을 정의한 다음 [Ctrl-w]를 사용한다.

5.8.4 본문의 이동과 복사

제거고리로부터 회복하는 방법으로 본문을 이동시킬수 있다. [Ctrl-y]는 일반적으로 제거고리의 본문(구역에서 지워 진 본문도)을 회복하는데 리용된다. [Ctrl-k][Ctrl-k]로 행을 지웠다면 다음의 건을 사용하여 새로운 위치에 그 행을 회복할수 있다.

```
[Ctrl-y]          본문을 되살리는 포괄적인 건
```

삭제된 본문은 새 위치에서 유표의 오른쪽에 배치된다. 이렇게 임의의 본문을 이동시킨다. [Alt-3][Alt-d]로 3개의 단어들을 지운 다음 [Ctrl-y]를 사용하면 그 단어들도 회복시킬수 있다. 그 본문자체가 별도의 행에 배치되도록 하여야 한다면 [Ctrl-y]를 사용하기 전에 행을 만들어야 한다.

본문을 복사하려면 구역을 리용하여야 한다. [Ctrl-w]가 구역안의 본문을 지운다면 본문을 복사하는 지령은 다음과 같다.

```
            표식                                        구역

"A programmer is a person who   passes   as an exacting expert on the   basis of
being able to turn out, after  innumerable punching,an infinite series   of incompr
ehensive answers calculated with micrometric precisions from vague assumption based
on debatable figures taken from inconclusive documents and carried out on instr
uments of problematical accuracy by  persons of dubious reliabilityand question
able mentality for the avowed purpose of annoying and confounding a hopelessly de
fenseless department that was unfortunate enough to ask for the information in t
he first place." -- IEEE Grid news magazine

        점                      [Ctrl-w]

"A programmer is a person who passes as an exacting expert on the basis of being
able to turn out,after innumerable punching,an infinite series of incomprehen
sive answers for the avowed purpose of annoying and confounding a hopelessly de
fenseless department that was unfortunate enough to ask for the information in t
he first place." -- IEEE Grid news magazine
```

그림 5-10. 구역안에서 본문의 삭제

[Alt-w]　　　먼저 구역을 정의한다

이 지령은 본문을 제거고리에 복사한다. 삭제된 본문의 경우와 같이 복사된 본문도 [Ctrl-y]를 리용하여 회복한다. 그 경우에도 본문은 유표의 오른쪽에 배치된다.

5.8.5 본문의 전위

사용자는 린접한 2개의 문자나 행을 아주 쉽게 교체할수 있다. 이를 위한 건은 [Ctrl-t]이다. 실수하여 Computer를 Compuetr로 틀리게 입력하였다면 유표를 t에 옮기고 다음의 건을 누른다.

[Ctrl-t]　　　et가 te로 된다

2개의 문장도 서로 위치바꾸기를 할수 있다. 유표를 아래의 행으로 옮기고 다음의 건을 사용한다.

[Ctrl-x][Ctrl-t]　　　vi에서의 ddP와 같다

본문의 전위(transposing)를 그림 5-11에 보여 준다.

```
They would be more convenient to use
The computer is giving way to hand-held devices

    [Ctrl-t]

They would be more convenient to use
The computer is giving way to hand-held devices

    [Ctrl-x][Ctrl-t]

The computer is giving way to hand-held devices
They would be more convenient to use
```

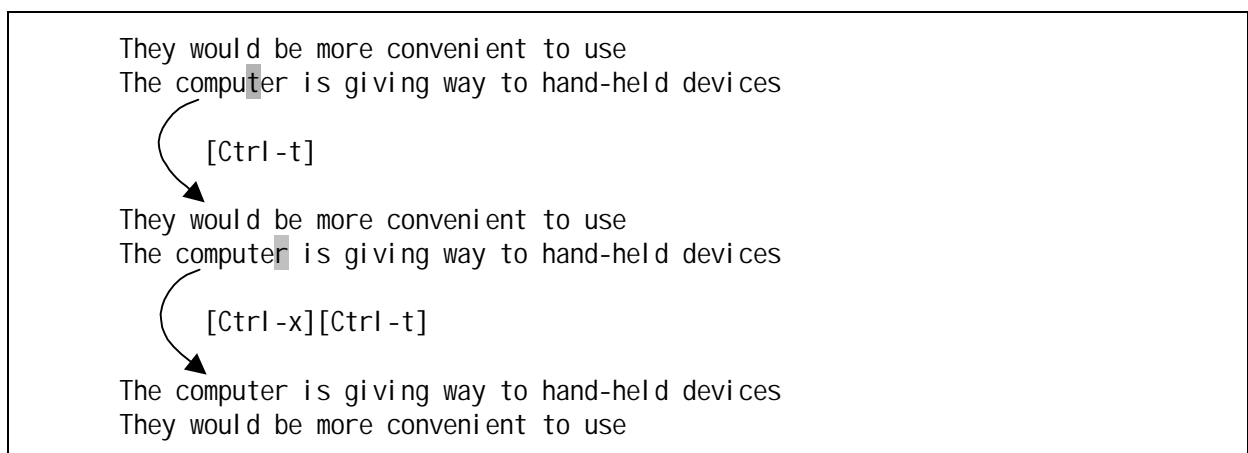



그림 5-11. 본문의 전위

주해 [Ctrl-d]로 지워 진 본문이 보관되지 않는다는 설명은 정확하지 않다. 모든 편집조작들은 보관되며 취소할수 있다. [Ctrl-d]가 삭제된 본문을 제거고리에 보내지 않는다는것은 옳다. 제거고리로부터의 본문회복에 대하여 배우면 이 점에 대하여 알게 될것이다.

5.9 본문의 대소문자변경

emacs는 본문의 대소문자를 변경하는 포괄적인 기능들을 가지고 있다. 그 기능들은 구역이나 단어들에서 다 사용될수 있다. 전체 단어를 대문자로 변환하려면 단어의 시작으로 가서 다음의 건을 누른다.

[Alt-u]　　　　전체 단어를 대문자로 만든다

이와 류사한 방법으로 단어를 소문자로 변환하려면 [Alt-l]을 사용하여야 한다. 한 문자를 대문자로 만들려면 그 문자에 유표를 배치한 다음 아래의 건을 사용한다.

[Alt-c]　　　　첫 문자를 대문자로 만든다

이 3개의 지령들은 모두 유표를 다음단어로 이동시키므로 이 건을 계속 누르고 있으면 50개 단어도 잠간사이에 변환할수 있다. 문자와 단어의 대소문자변환을 그림 5-12에 보여 준다.

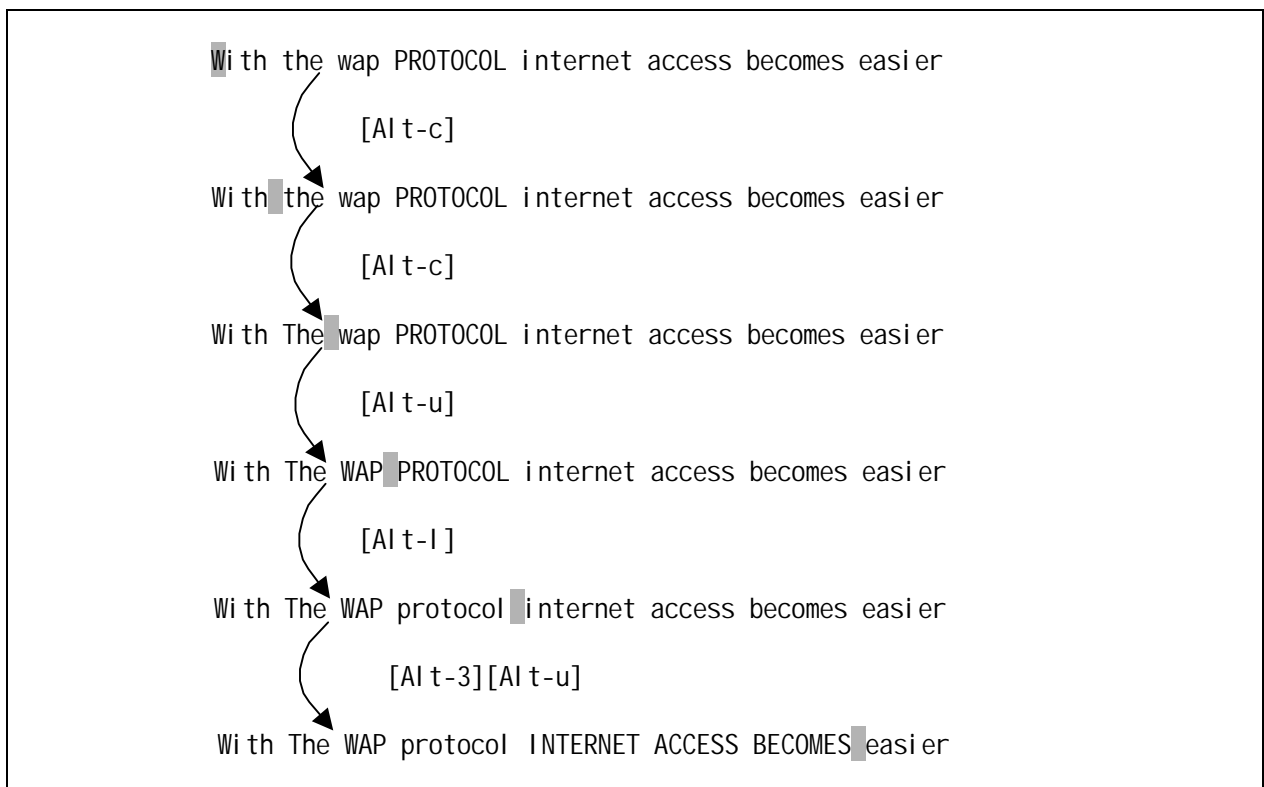



그림 5-12. 본문의 대소문자변경

큰 본문블로크의 대소문자를 변환하려면 구역을 사용하여야 한다. 구역안의 본문을 변환하기 위한 지령들은 다음과 같다.

[Ctrl-x][Ctrl-u]　　　　구역안의 전체 본문을 대문자로 변환

[Ctrl-x][Ctrl-l]　　　　구역안의 전체 본문을 소문자로 변환

이 건들은 단어에서 사용될 때와 전혀 다르다. 이러한 차이점은 이 지령들을 기억하기 힘들게 한다.

참고

구역에서의 대소문자변환기능이 불가능상태로 되여 있을수도 있다. 그때 emacs는 이 기능을 항구적으로 가능하게 하겠는가를 문의한다. 이것은 upcase-region과 downcase-region지령들에 의하여 조종된다. 구역을 대문자로 변환할 때의 최종요구에 y로 대답하면 emacs는 .emacs안에 아래와 같은 설정을 만든다(down-case지령에 대해서도 류사하다).

```
(put 'upcase-region 'disabled nil)
```

5.10 지령완성하기기능

앞에서 우리는 emacs의 2개의 지령들인 line-number-mode와 goto-line의 완전한 본문을 입력하였다. **지령완성하기**(command completion)기능이 어떻게 이 타자작업의 일부를 줄일수 있는가를 보기로 하자.

이 방법의 원리는 다음과 같다. 요구하는 정도의 지령본문을 입력하고 [Tab]건을 누른다. emacs는 지령을 완성하여 줄수도 있고 입력된 문자렬을 포함하는 모든 지령들의 목록을 보여 줄수도 있다. 몇개의 문자들을 더 입력하고 다시 [Tab]를 누른다. 이 과정은 emacs가 목록에서 완성된 지령문자렬을 찾아낼수 있을 때까지 계속된다.

goto-line지령을 고찰하여 보자. 먼저 모든 emacs지령의 앞에 사용되는 건인 [Alt-x]를 누른다. 문자렬 M-x가 나타나면 문자렬 go를 입력하고 [Tab]를 누른다.

M-x **go**[Tab]

그러면 새 창문이 열리고 emacs는 문자렬 go가 포함되는 6개의 문자렬을 보여 준다. 그림 5-13에 완성목록을 보여 준다.

```
In this buffer, type RET to select the competion near point.

Possible completions are:
gomoku                              goto-address
goto-address-at-mouse              goto-address-at-point
goto-char                           goto-line
… blank lines …
--1-:--F1    *Completions*      (Completion List)--L1--All-----------------
M-x go
```

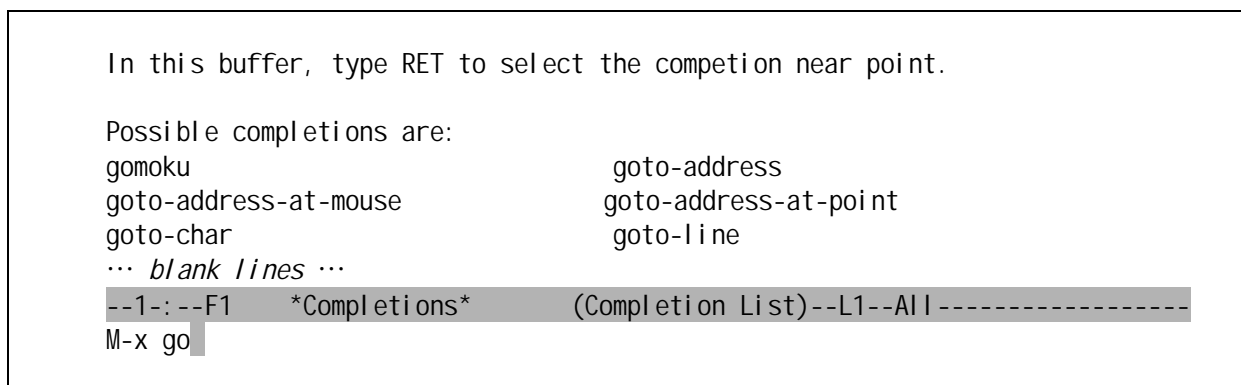



그림 5-13. 지령완성화면

이 목록을 주의 깊게 관찰하여보자. 만일 사용자가 현재의 부분문자렬에 t를 덧붙이고(got로 만들고) 다시 [Tab]를 누르면 emacs는 그것을 goto-로 늘일것이다.

M-x **got**[Tab]   goto－로 된다

이것은 5개의 문자렬들과 대응된다. 이때 goto-line과 대응시키려면 1을 입력해야 한다.

M-x **goto-1**[Tab]   goto-line으로 된다

그러면 소형완충기에 완성된 지령이름이 나타날것이다. 지령은 완성되였으며 사용자는 [enter]건을 누르고 goto-line지령이 요구하는 행번호를 입력할수 있다. 요약하면 emacs가 지령을 완성하도록 하기 위하여서는 문자렬이 유일한것으로 될 때까지 문자를 입력하고 [Tab]를 눌러야 한다는것이다.

특정한 행으로 이동하려면 문자를 더 입력해야 한다. 다시 말하여 완성하기기능이 큰 도움을 주지 않는다. 그러나 line-number-mode지령에서는 매우 효과적이다. 문자렬 lin을 입력하기만 하면 이 작업이 수행된다.

```
M-x lin[Tab]            line-number-mode로 된다
```

[Tab]를 누를 때 emacs가 현시하는 목록들을 보면 특정한 문자렬로 시작되는 지령들을 알수 있다. 그러면 완성하기기능을 충분히 사용하기 위하여 지령을 얼마나 입력하여야 하는가를 알게 될것이다. 종전의 실례에서는 최소의 건조작으로 goto-line문자렬을 완성하기 위해서는 got[Tab]l[Tab]를 입력하여야 한다는것을 말해 준다. 표 5-5에 몇가지 emacs지령들에 대한 지령완성하기기능의 사용법을 보여 준다.

표 5-5. emacs에서 지령완성의 실례

입력된 문자렬	확장된 문자렬
got[Tab]	goto-
goto-l[Tab]	goto-line
lin[Tab]	line-number-mode
replace-s[Tab]	replace-string
repl[Tab]r[Tab]	replace-regexp
que[Tab]	query-replace
que[Tab]-[Tab]	query-replace-regexp
call[Tab]	call-last-kbd-macro

5.11 편집의 취소와 재실행

다른 편집기들과 마찬가지로 emacs도 수행된 편집조작을 뒤집을수 있는 취소(undo)기능과 재실행(redo)기능을 가지고 있다. 이와 대조적으로 vi(vim이 아니라)는 마지막편집조작만을 취소할수 있다. emacs문서에 의하면 그 기능은 두개의 건누름렬이 할당되면 리용할수 있으며 충분히 가치가 있다. 실제로는 3가지 방법이 있다.

```
[Ctrl-x]u
[Ctrl-_]        밑선을 쓰자면 [Shift]를 눌러야 한다
[Ctrl--]        이음표만으로 되여 있다
```

실수하여 완충기를 변경했을 때에는 그 건조합들중 하나를 누르면 된다. 첫번째것은 거의 모든 책들에서 서술하고 있지만 다른 두 지령이 사용하기가 더 쉽다. 그러나 그 건조합들을 사용하기전에 그 건조합이 체계에서 동작하는가를 검사하여야 한다. PC를 사용하고 있는 경우에 기호 -와 _는 같은 건이며 밑선은 [Shift]를 사용하여 호출한다. [Ctrl--]는 [Shift]를 요구하지 않으므로 제일 사용하기 쉽다.

완충기에 만들어 진 모든 단일한 변경이 취소될 때까지 이러한 건조작들중의 어느 하나(례하면 [Ctrl--])를 반복적으로 계속 누를수 있다. 최종적으로는 방식행에서 두개의 별표가 없어 지는것을 보게 된다. 이것은 파일을 마지막으로 보관하였을 때의 완충기의 상태이다. [Ctrl--]를 계속 누르면 마지막보관전에 만들어 진 변경들도 취소될것이다. 최종적으로 체계는 다음과 같은 통보를 사용자에게 알려 줄수 있다.

```
No further undo information
```

[Ctrl--]를 다시 누르면 어떻게 되겠는가? 여기에 특이한 문제점이 있다. emacs는 사용자가 취소했던 모든 조작을 재실행하기 시작한다. 이런 방법으로 사용자는 본래상태로 돌아 갈수 있다. 이 련속적인 동작은 끝없이 계속된다. 그러나 이 재실행기능은 문서화가 잘되여 있지 않다.

주해

모든 편집지령들을 무제한 취소할수는 없다. emacs변수 undo-limit가 그 한계값을 조종한다. 기정적으로 한계값은 20,000byte의 취소정보로 설정되는데 이것은 대부분의 목적에 아주 적당하다. 취소정보가 이 한계값을 초과하면 페품수집(garbage collection)이 FIFO형식으로 동작하며 제일 오래된 지령들은 버린다.

5.12 문자렬탐색

emacs는 단순한것으로부터 복잡한것에 이르기까지 여러가지의 패턴정합방법들을 가지고 있으며 이 방법은 vi에 비길수 없이 정교하다. 탐색은 정규식뿐아니라 간단한 문자렬에 의해서도 진행된다. 이 절에서는 문자렬탐색에 대하여 고찰하며 두가지 탐색기술 즉 증가탐색과 비증가탐색을 사용한다.

5.12.1 증가탐색([Ctrl-s]와 [Ctrl-r])

이 방법에서 emacs는 첫 문자가 입력되는 순간에 탐색을 시작하며 문자의 첫 실체가 발견되는 곳에 유표를 배치한다. 그다음의 문자들은 부분문자렬을 이루며 다시 탐색된다. 탐색을 증가하는 이 방법은 완전한 문자렬이 입력될 때까지 계속된다. **증가탐색**(incremental search)은 [Ctrl-s]와 그뒤에 탐색문자렬을 입력하는 방법으로 호출된다.

단어 mail을 탐색하는 과정을 보기로 하자. 먼저 [Ctrl-s]를 눌러야 하며 그때 emacs는 소형완충기를 통하여 탐색문자렬을 요구한다. 문자렬의 첫 문자 m을 입력하자.

```
I_search: m
```

탐색은 즉시에 시작된다. 유표는 처음 나타나는 m(단어 moon이 될수도 있다.)에 옮겨 진다. 다음 a를 입력하면 machine으로 이행할수 있다. i를 누르면 maintain으로 옮겨 갈수 있다. 최종적으로 파일에 문자렬 mail이 있다면 유표는 거기로 이동한다. 그러면 사용자는 [Enter]를 눌러 탐색문자렬의 완료를 표시하여야 한다.

증가탐색은 많은 우점을 가지고 있다.

- 다른 편집기에서 쓰이는 방법들에 비하여 탐색이 빠르다.
- emacs는 사용자가 입력한 문자렬이 파일에 존재하는가를 즉시 알려 준다. 만일 quantum을 찾으면서 qu대신에 qa를 입력하면 파일에는 그러한 문자렬이 없기때문에(q로 시작되는 단어는 뒤에 항상 u가 놓인다.) qu를 입력하자마자 emacs는 즉시에 탐색을 거절한다.
- 문자렬에 타자오유가 있을 때 후진건을 리용하면 오유가 지워 지면서 유표를 종전의 위치로 돌려 보낸다. 이것은 후진건을 쓸 때마다 련속적으로 수행되며 문자렬을 완전히 지웠을 때에는 탐색을 시작한 점으로 되돌아 오게 된다.

파일끝에 위치하고 있을 때에는 탐색을 뒤로 진행할 필요가 있다. 그러한 경우에는 [Ctrl-s]가 아니라 [Ctrl-r]를 사용하여야 한다. 모든것이 다 그러하지만 문자렬을 찾아 낼 때 [Enter]를 누르는것을 잊지 말아야 한다. 그림 5-14에 증가탐색에 대하여 보여 준다.

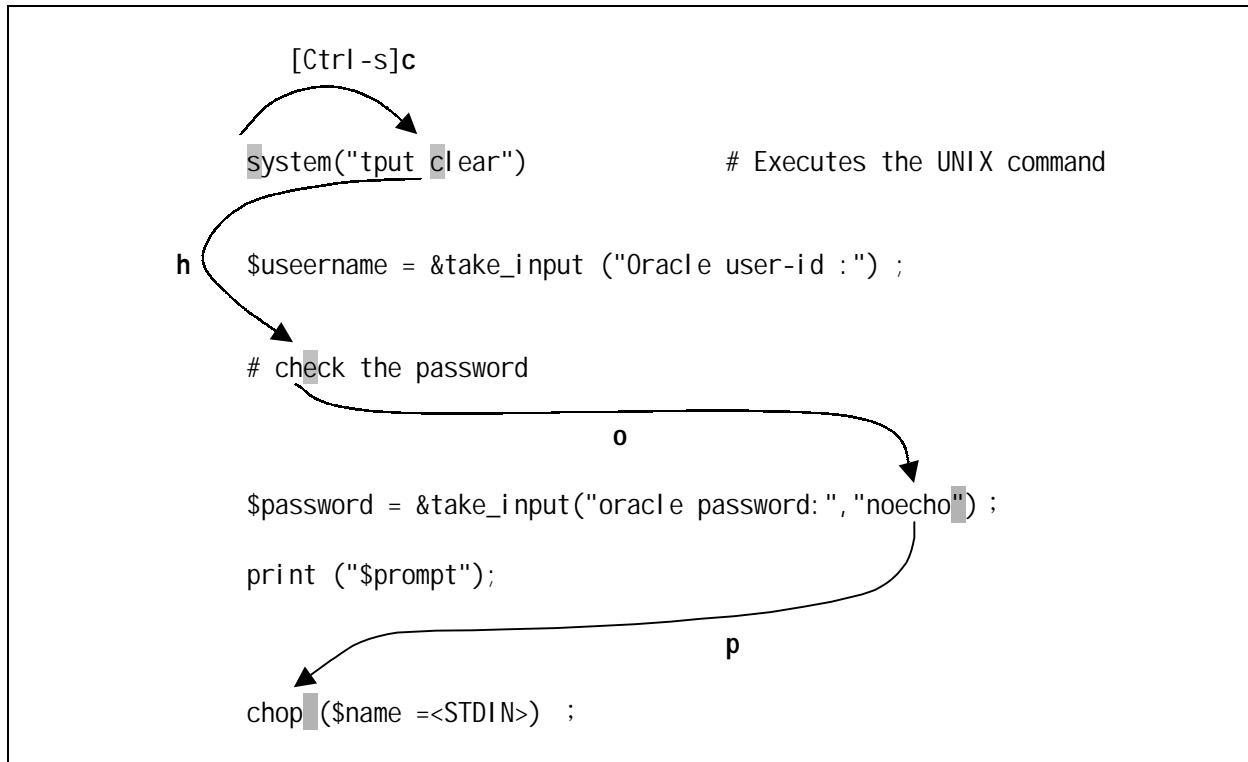



그림 5-14. 문자렬 chop에 대한 증가탐색

5.12.2 마지막증가탐색의 반복([Ctrl-s]와 [Ctrl-r])

탐색을 반복하려면 동일한 지령 [Ctrl-s]와 [Ctrl-r]를 문자렬없이 사용하여야 한다. 처음 반복할 때는 이 건을 두번 눌러야 한다. 파일의 시작이나 끝에 위치하고 있는 경우에는 emacs가 다음과 같은 통보문을 내보낸다.

```
Failing I-search: quantum
```

감돌기기능(4.14.1)은 emacs에서도 적용된다. 다음번에 [Ctrl-s]를 누르면 그 끝을 감돌아 탐색을 파일의 시작점으로부터 다시 시작한다. 반대방향의 탐색에서도 꼭같이 적용된다.

주의

임의의 시각에 탐색을 취소하려 할 때에는 [Esc]나 [Enter]건을 눌러야 한다. 입력된 문자렬을 Backspace로 지워 버린 다음 본문을 입력하면 그 본문은 여전히 완충기가 아니라 탐색문자렬에 추가될것이다. 많은 경우 emacs는 경보음을 울리면서 우에서 보여 준것과 같은 오유통보문을 내보낼것이다. 탐색방식에서 [Esc]나 [Enter]는 이 동작을 취소하고 보통방식으로 돌아 가게 한다.

5.12.3 비증가탐색

emacs에서는 다른 편집기들에서 쓰이는것과 같은 단순한 탐색방법 즉 **비증가탐색**(nonincremental search)도 선택할수도 있다. 처음에는 같은 건([Ctrl-s]나 [Ctrl-r])을 사용하지만 탐색패턴을 요구할 때에는 [Enter]를 눌러야 한다.

```
I-Search:[Enter]
```

그러면 emacs는 문자렬을 요구한다.

```
Search: quantum[Enter]
```

종전의 탐색을 비증가적방법으로 수행하는 실제적인 방법을 아래에 보여 준다.

```
[Ctrl-s][Enter]quantum[Enter]
```

증가탐색을 반복할 때와 같은 방법으로 비증가탐색도 반복할수 있으며 여기에는 특별히 고려할 문제가 없다. 탐색 및 반복지령들을 표 5-6에 보여 주었으며 그림 5-15에는 비증가탐색의 실례를 보여 준다.

표 5-6. emacs에서 탐색 및 반복지령

지 령	기 능
[Ctrl-s]pat	패턴 pat에 대한 전진방향의 증가탐색
[Ctrl-r]pat	패턴 pat에 대한 후진방향의 증가탐색
[Ctrl-s][Enter]pat	패턴 pat에 대한 전진방향의 비증가탐색
[Ctrl-r][Enter]pat	패턴 pat에 대한 후진방향의 비증가탐색
[Ctrl-s]	전진방향에서 탐색을 반복(증가 및 비증가)
[Ctrl-r]	후진방향에서 탐색을 반복(증가 및 비증가)
[Esc]	탐색을 취소
[Ctrl][Alt]spat	정규식 pat에 대한 전진방향의 증가탐색(후진방향인 경우에는 r를 리용)
[Ctrl][Alt]s[Enter]pat	정규식 pat에 대한 전진방향의 비증가탐색(후진방향인 경우에는 r를 리용)
[Ctrl][Alt]s[Enter][Enter]	정규식에 대한 전진방향탐색을 반복(증가 및 비증가)
[Ctrl][Alt]r[Enter][Enter]	정규식에 대한 후진방향탐색을 반복(증가 및 비증가)

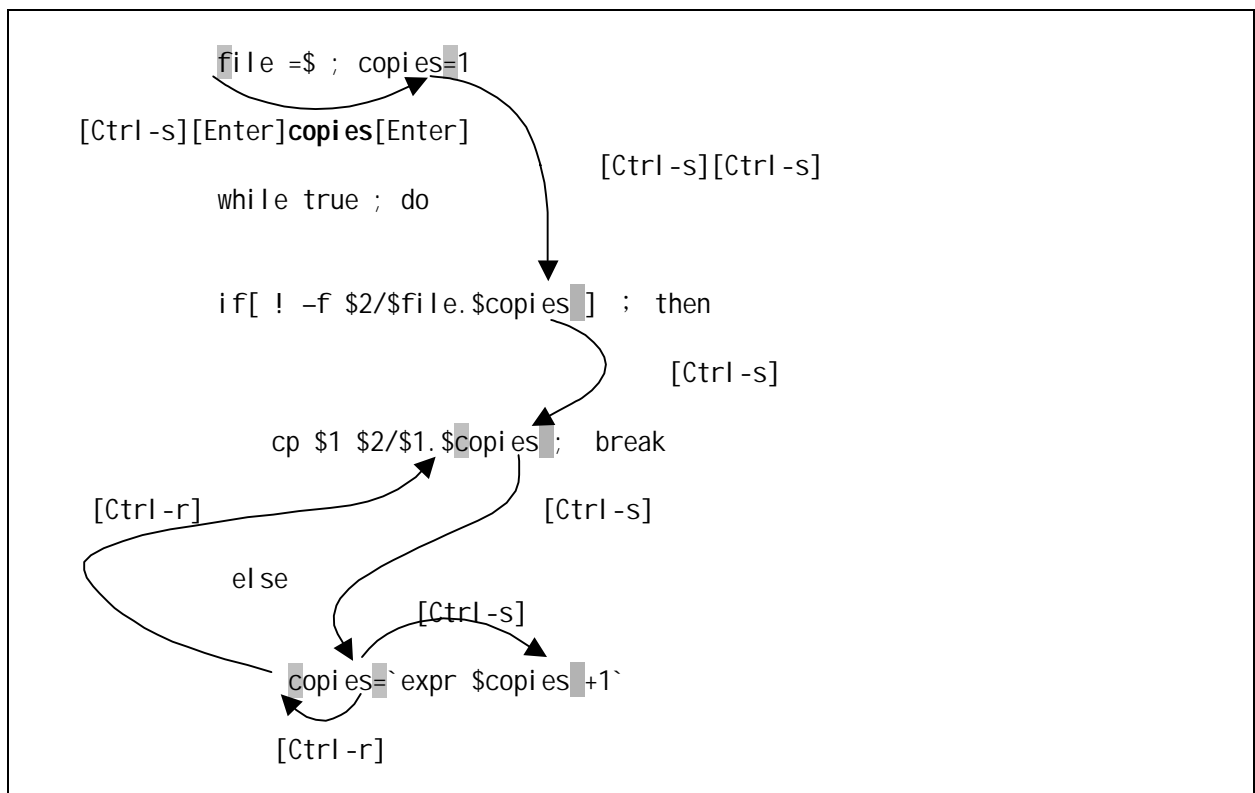



그림 5-15. 문자렬 copies에 대한 비증가탐색

참고

우연히 탐색하려는 문자렬과 같은 어떤 단어의 가까이에 있는 경우에는 문자렬을 입력할 필요가 없다. 먼저 단어앞의 공백에 유표를 옮기고 [Ctrl-s]를 누른다. 그다음 문자렬을 입력할 대신 [Ctrl-w]를 누른다. 그러면 그 단어가 소형완충기에 복사되며 [Ctrl-s]를 사용하여 탐색을 반복할수 있다.

5.13 정규식을 리용한 탐색

vi와 마찬가지로 emacs는 류사한 문자렬들을 찾기 위하여 몇개의 특수한 문자들로 이루어 진 패턴을 사용한다. 이 패턴을 정규식(regular expression)이라고 하며 이 책에서 취급하는 중요한 문제의 하나이다. 이 내용은 4장에서 소개되였으며 vi사용자가 아닌 경우에도 그 부분(4.15)을 읽어 보아야 할것이다. 왜냐하면 거기서 설명된 모든 개념들이 emacs에도 적용되기때문이다.

4.15에서 설명되였던 내용을 다시 상기하여 보면 아래의 표현을 리용하여 michael과 michel에 대한 탐색을 진행할수 있다.

```
micha*el        a*는 a, aa, aaa, 지어 a가 없는 경우와도 일치한다
```

여기서 *는 앞의 문자가 임의의 회수만큼 발생하거나 전혀 발생하지 않을수도 있다는것을 가리킨다. micha*el은 여러개의 문자렬들을 정합할수 있는 정규식이다.

또한 어떤 모임안에서 한개의 문자를 지정하기 위한 **문자모임**(character class)도 사용할수 있다. 이것은 [iy]가 i 또는 y와 일치되는 한문자정규식이라는것을 의미한다. christie 와 christy에 일치되는 패턴은 다음과 같이 될것이다.

```
christ[iy]e*        [iy]는 i 또는 y를 의미한다
```

행의 시작과 끝에 있는 표현들과 정합하기 위하여서는 정착문자(anchoring character) ^와 $를 각각 리용할수 있다. 아래의 표현은 C프로그람에서 설명행을 찾아 낸다.

```
^#        ^는 시작점에 일치한다
```

점(.)은 한 문자를 의미하며 따라서 두 점으로 된 패턴은 두개의 문자와 일치한다. 그러나 그 점이 *와 결합될 때에는(.*에서와 같이) 완전히 다른 의미 즉 임의의 수의 문자를 표현한다. 문자렬 You 또는 you를 포함하는 echo지령을 찾아 보려면 패턴 echo.*[yY]ou를 사용한다.

표 5-7. emacs에서 사용되는 정규식문자들

기 호	의 미
*	앞문자의 령이상의 발생과 일치한다
[pqr]	p, q, r중의 어느 한 문자에 일치한다
[^pqr]	p, q, r가 아닌 한 문자에 일치한다
.	한개의 문자에 일치한다
^pat	행의 시작에 있는 패턴 pat와 일치한다
pat$	행의 끝에 있는 패턴 pat와 일치한다
\<pat	단어의 시작에 있는 패턴 pat와 일치한다
pat\>	단어의 끝에 있는 패턴 pat와 일치한다
\bpat	단어의 시작에 있는 패턴 pat와 일치한다
pat\b	단어의 끝에 있는 패턴 pat와 일치한다
\Bpat	임의의 곳에서 단어의 시작에 있는 패턴 pat와 일치한다
pat\B	임의의 곳에서 단어의 끝에 있는 패턴 pat와 일치한다

emacs는 일부 특수한 문자들을 가지고 있다. \b는 그것이 배치되는 자리에 따라 단어의 시작 또는 끝에 있는 패턴과 정합한다.

```
\bfoot          lightfoot와는 일치하지 않는다
wood\b          woodcock와 일치하지 않는다
```

처음것은 michael foot와는 일치하지만 foot가 들어 있는 lightfoot와는 일치하지 않는다. 두번째것은 wood는 선택하지만 woodcock는 버린다. 이와 류사하게 \B는 \b를 반전한다. 즉 그것은 비단어경계(단어의 시작도 끝도 아닌 곳)에 정합된다.

정규식문자들은 15장과 16장, 20장에서 상세하게 설명되였다. grep나 sed, awk, perl과 같은 강력한 도구들이 이 표현들을 리용하기때문에 이것은 UNIX에서 아주 중요하다. 치환을 수행할 때에도 그것들이 아주 쓸모 있다는것을 알게 될것이다.

탐색기술

이제 탐색에서 정규식들에 대한 지식을 사용하여 보자. 이 탐색들은 증가탐색일수도 있고 비증가탐색일수도 있다. 증가방식으로 전진탐색하기 위하여서는 아래의 건들을 사용한다.

```
[Ctrl][Alt]s            후진탐색을 위해서는 r
```

emacs는 소형완충기에서 민활하게(재빠르게) 응답한다.

```
Regexp I-search:        I는 증가를 의미한다
```

이때 사용자는 방금 론의한 임의의 표현식들을 입력하고 [Enter]를 입력할수 있다. 또한 여기에 단순한 문자렬도 지정할수 있다. 유표는 정규식의 매 문자가 입력되는데 따라 그 표현과 일치되는 첫 발생에로 점차적으로 이동한다. 입력을 완성한후에는 [Enter]건을 누르는것을 잊지 말아야 한다.

전진방향의 비증가탐색은 [Ctrl][Alt]s[Enter]로 시작한다. 정규식을 입력하고 다시 한번 [Enter]를 누른다. 즉 행의 시작과 끝에 각각 문자렬 dnl을 포함하고 있는 모든 행들을 찾아 내기 위해서는 다음의 방법으로 탐색을 진행하는것이 필요할것이다.

```
[Ctrl][Alt]s[Enter]^dnl.*dnl$[Enter]
```

emacs는 이 표현식을 간단하게 만들수도 있지만 그것은 별개의 문제이다.

정규식탐색을 반복하는것은 매우 시끄러운 작업이다. 즉 표현을 입력할 때까지 같은 건들을 사용하는데 이때에는 또 다른 [Enter]건을 추가한다.

```
[Ctrl][Alt]s[Enter][Enter]
```

더우기 파일의 끝점을 만날 때 그 탐색은 감돌지 않는다. 이렇게 하면 n과 N을 사용하여 모든 형태의 탐색을 반복할수 있으며 감돌기는 결코 문제로 되지 않는다.

5.14 탐색과 치환

emacs는 대화형과 비대화형을 둘 다 가진 강력한 탐색 및 치환능력을 제공한다. vi와 emacs가 다른것들보다 앞선 수단으로 되는 리유는 그것들이 다 정규식을 사용하여 류사한 한 묶음의 문자렬들을 간결한 하나의 표현으로 치환한다는것이다. 치환지령들을 표 5-8에 보여 준다.

5.14.1 비대화형치환

대화없이 전역적으로 단어 Internet를 World Wide Web로 치환하려면 먼저 [Alt-x]를 누른다.

표 5-8. emacs에서 치환지령

지 령	기 능
[Alt-x]replace-string	비대화형으로 문자렬을 치환한다
[Alt-x]replace-regexp	비대화형으로 정규식을 치환한다
[Alt-%]	대화형으로 문자렬을 치환한다([shift]를 요구)
[Alt-x]query-replace	대화형으로 문자렬을 치환한다
[Alt-x]query-replace-regexp	대화형으로 정규식을 치환한다

이 건은 emacs지령을 입력하기 위하여 사용된다. 이제는 replace-string지령을 입력한다.

```
M-x replace-string[Enter]
```

지령완성하기기능은 [Alt-x]repl[Tab]s[Tab]를 사용할것을 요구한다. 사용자에게 치환될 문자렬을 입력할것을 재촉한다.

```
Replace string: Internet[Enter]
```

이번에는 emacs가 목적문자렬을 요구한다.

```
Replace string Internet with: World Wide Web[Enter]
```

이때 치환은 완충기를 통하여 진행된다. 만일 기대한대로 진행되지 않았다고 느껴 지면 [Ctrl-_] 또는 [Ctrl--]로 그 조작을 취소한다.

비대화식으로 정규식을 치환하기 위하여서는 replace-string지령의 자리에 replace-regexp를 사용하여야 하며 나머지는 모두 같다.

5.14.2 대화형치환

동일한 이름의 emacs지령을 사용하는 **질문치환**(query-replace)기능을 리용하면 대화적방식으로 문자렬들을 치환할수 있다. 이때 우리는 [Alt-x]를 사용하지 않을것이다. 왜냐면 건맺기 [Alt-%]가 그 과제를 위해서 유용하기때문이다. 우리가 이 조작순서를 사용할 때 체계는 query-replace지령을 실행하며 2개의 문자렬을 요구한다.

```
Query replace: Internet[Enter]
Query replace Internet with: World Wide Web[Enter]
```

그다음 그것은 처음 나타나는 Internet로 이동하여 아래의 질문을 보낸다.

```
Query replacing Internet with World Wide Web: (? for help)
```

여기서 사용자가 가질수 있는 10개의 선택항목이 있으나 대부분의 경우에는 아래의 선택항목들만이 필요할것이다.

y 또는 [Spacebar] - 현재의 발생을 치환하고 다음의것으로 이동한다.

n - 치환하지 않고 다음의것에로 뛰여 넘는다.

q 또는 [Enter] - 치환없이 탈퇴한다.

. - 이 실체만을 치환하고 탈퇴한다.

! - 물어 보지 않고 나머지발생들을 모두 치환한다.

완충기를 보관하지 않고 탈퇴하려고 할 때 emacs는 류사한 선택항목들을 제공한다. 그러나 그것들은 여기서 다른 의미를 가진다. 필요한 모든 치환들이 수행되였다고 느껴 질 때 q를 사용하여 그 치환조작을 중지할수 있다. 한편 이미 여러차례의 치환들을 만들었고 나머지를 위한 더이상의 확증이 요구되지 않는다고 느껴 질수도 있다. 그러한 경우에는 !를 사용하여 emacs가 나머지치환들을 완성하게 한다. 이 기능들은 vi에서는 리용할수 없다.

emacs는 정규식을 대화식으로 치환하기 위하여 query-replace-regexp지령을 사용한다. 그것은 또한 치환조작의 도중에 매우 쓸모 있는 과제 즉 **재귀편집**(recursive editing)을 수행할수 있다. 이것은 사용자가 작업도중에 림시적으로 벗어 나 작은 편집과제를 수행하는것을 허용한다.

5.15 다중파일과 창문, 완충기의 사용

사용자는 파일을 열 때 완충기를 그 파일과 련결시킨다. 이 련계는 간단한 보관조작으로 완충기를 디스크파일에 보관할수 있게 한다. 사용자는 한개이상의 창문들에서 완충기를 그 파일의 화상으로서 현시할수도 있지만 완충기와 창문사이에는 그러한 관계가 없다. 창문은 단순히 그 완충기의 열람기로서 동작한다. 창문을 닫을 때 완충기는 여전히 열려 있으며 아무때나 다시 호출될수 있다. 완충기와 창문들사이에 차이점은 미묘하지만 명백하며 아래의 내용들이 그 차이를 명백히 하는데 도움을 줄것이다.

5.15.1 창문의 조종

vi(vim은 아니다.)와 달리 emacs는 여러개의 창문들을 가지고 작업할수 있게 한다. 사용자는 화면을 요구하는 정도의 많은 창문들로 분할할수 있다. 창문은 임의의것을 포함할수 있다. 그것들은 비여 있을수도 있으며 동일한 파일이나 서로 다른 파일들을 포함할수도 있다. 우리는 대체로 두개의 창문을 편리하게 여기므로 두개의 창문을 가지고 작업하도록 할것이다.

2개의 분리된 창문들에 동일한 파일을 현시하려면 아래의 건조작을 사용한다.

[Ctrl-x]2	2개의 창문으로 분할한다

```
Buffers Files Tools Edit Search Mule Insert Help
option=-e
while echo $option "Designation code: \c"
do
      read desig
          case "$desig" in
      [0-9][0-9]) if grep "^$desig" desig.lst >/dev/null # if code exists
                   then echo "Code exists"
--1-:--F1  dentry1.sh       (Shell-script Abbrev)--L1--Top----------------------
             case "$desig" in
      [0-9][0-9]) if grep "^$desig" desig.lst >/dev/null # If code exists
                   then echo "Code exists"
                        continue
                        fi ;;
               *)  echo "Invalid code" ; continue ;;
              esac
--1-;--F1  dentry1.sh       (Shell-script Abbrev)--L1--Top----------------------
```

그림 5-16. 분할된 창문

그러면 화면이 2개로 분할되는것을 보게 될것이다. 그림 5-16의 분할된 창문들은 동일한 파일의 서로 다른 토막들을 조금씩 보여 준다. 이런 방법으로 사용자는 다른 창문에서 본문을 흘러 보냄으로써 프로그람의 다른 부분들을 볼수 있다. 두개의 창문이 동일한 파일을 보여 주고 있으므로 한 창문에서 수행한 편집변경이 다른 창문에서도 나타난다.

사용자들은 흔히 2개의 창문에서 서로 다른 2개의 파일을 편집하게 될것이다. 다른 창문으로 이동하기 위한 지령은 다음과 같다.

[Ctrl-2]o	다른 창문으로

이 지령을 줄 때마다 유표는 두 창문사이를 왕복한다. 만일 3개의 창문을 가지고 있다면 매 창문들을 순차적으로 이동하게 될것이다. o는 other를 기억하기 위한것이며 사용자를 다른 창문에 데려다 준다.

만일 한 창문이 더 큰 화면을 요구하면 창문의 수직크기를 늘일수 있다.

[Ctrl-x]^	창문의 크기를 늘인다

창문의 크기를 줄이는것은 불편하다. 2개의 창문을 가지고 작업할 때에는 [Ctrl-x]o를 리용하여 다른 창문에 이동한 다음 그 창문의 크기를 늘이는것이 더 쉽다.

이제 한 창문에서 다른 파일을 열어 보자. 아래의 지령은 이미 알고 있을것이다.

[Ctrl-x][Ctrl-f]	vi에서 :e foo와 같다

다른 파일이름을 입력한후에는 2개의 창문에 서로 다른 2개의 파일을 가지게 될것이다. 더우기 2개의 완충기들도 가질것이다. 이제 이 지령으로 잘못된 파일을 열었다고 가정하자. 이 경우에 같은 지령을 또다시 사용하겠는가? 아니다. 만일 이 파일이 전혀 필요하지 않다면 그것을 다른 파일로 교체해야 한다. 사용해야 할 지령은 다음과 같다.

[Ctrl-x][Ctrl-v]	현재의 완충기를 바꾼다

이것은 [Ctrl-x][Ctrl-f]를 가지고 틀리게 가져 온 현재의 완충기를 제거한다. 그렇치 않고 [Ctrl-x][Ctrl-f]를 다시 사용하면 여전히 틀리게 호출하였던 종전파일의 완충기에 가게 된다. vi와 달리 emacs는 또 다른 파일을 호출하기전에는 현재완충기를 보관하지도 않고 제거하지도 않으며 따라서 이 지령을 사용하지 않는다면 완충기목록에 그것들을 여러개 가질수 있다.

주해 지령 [Ctrl-x][Ctrl-f]와 [Ctrl-x][Ctrl-v]는 한개의 창문에서 작업할 때(일반적인 emacs방식)에도 두번째 파일을 열기 위해서 사용된다.

때로는 림시적으로나마 어떤 창문을 없애려고 할수도 있다. 화면상에서 현재의 창문을 유일한 창문으로 만들고 다른 모든 창문들을 닫아 버리려면 다음과 같은 지령을 사용한다.

[Ctrl-x]1	다른 모든 창문을 제거한다

반대로 현재의 창문을 없애고 다른 창문에로 이동할수도 있다.

[Ctrl-x]0	이 창문을 없앤다

마지막 2개의 지령들은 중요한 의미를 가진다. 창문을 없앤다고 하여 창문에 현시되였던 완충기를 없애지는 못한다. 즉 그것은 완충기목록안에 남아 있다. 창문들을 조작하기 위하여 사용되는 지령들을 표 5-9에서 보여 준다.

표 5-9. emacs에서 창문조종지령들

지 령	기 능
[Ctrl-x]2	현재창문을 두개의 창문으로 가른다
[Ctrl-x]o	창문들사이에서 이동한다
[Ctrl-x]^	현재창문의 수직크기를 늘인다([Shift]를 요구)
[Ctrl-x]1	다른 모든 창문들을 없애고 이 창문을 유일한 창문으로 만든다
[Ctrl-x]0	이 창문만을 없앤다
[Ctrl][Alt]v	다른 창문안의 본문을 앞으로 흘러 보낸다(뒤로 흘러 보내는 기능은 없다)

참고

[Ctrl][Alt]v를 사용하면 다른 창문에로 이동함이 없이 그 창문을 앞으로 흘러 보낼수 있다. 그러나 뒤로는 흘러 보낼수 없다. 만일 다른 창문을 제거해야 한다면 [Ctrl-x]1을 사용한다.

5.15.2 완충기의 조종

우리는 시종일관 완충기들을 조종하여 왔다. emacs는 완충기목록을 보여 주며 이 목록으로부터 완충기를 선택할수 있게 하는 여러개의 지령들을 제공한다. 아래의 지령을 사용하면 보존되지 않은 마지막상태에 있는 임의의 완충기를 불러 낼수 있다.

[Ctrl-x]b 다른 완충기를 호출한다

이 지령이 사용될 때 그것은 기정완충기로서 그 창문에서 편집되였던 마지막완충기이름을 제공한다. 마지막으로 편집된 파일에로 돌아 가려고 하는 경우에는 간단히 [Enter]건을 누른다. 2개의 파일을 번갈아 편집하고 있을 때 이 방법을 써서 현재파일과 방금전의 파일사이에서 절환할수 있다.

편집하려고 하는 완충기가 완충기목록에서 보여 준 기정완충기가 아닐 때에는 그 이름을 자체로 입력하거나 목록을 현시하고 거기서 선택할수 있다. [Ctrl-x]b를 입력하고 프롬프트가 나타날 때 [Tab]를 누른다. 대표적인 완충기목록을 그림 5-17에 보여 준다.

```
In this buffer, type RET to select the completion near point

Possible completions are:
*Messages*                           *scratch*
convert.sh                            dentry1.sh


--1-;--F1    *Completions*      (CompletionListAbbrev)--L1--All------------------
Switch to buffer: (default *scratch*
```

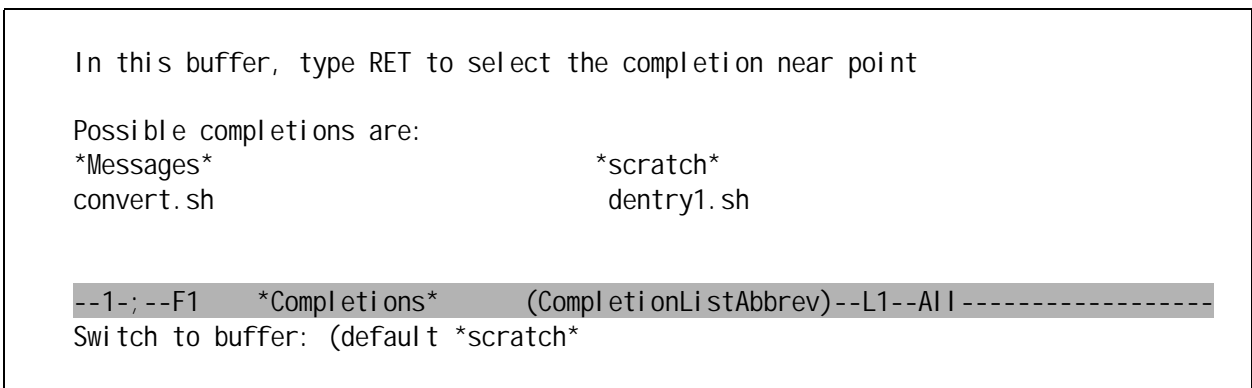



그림 5-17. [Ctrl-x]b를 리용한 완충기목록의 현시

그림의 완충기목록은 *scratch*완충기를 보여 주고 있는데 emacs는 사용자가 주해를 만들수 있도록 항상 그 완충기를 유용하게 한다. [Ctrl-x]o(필요하다면 두번)를 사용하여 완충기목록에서 사용자자신의 위치를 정할수 있다. 그다음 유표를 이동하고 [Enter]건을 누름으로써 임의의 완충기를 선택할수 있다. [Ctrl-x]0을 사용하여 이 창문을 닫거나 [Ctrl-g]를 리용하여 전반동작을 취소할수 있다.

완충기목록을 현시하기 위한 별도의 지령이 있다. 그것은 완충기로부터 또 다른 완충기에로 절환하

기 위한 지령과 약간의 차이가 있다.

```
[Ctrl-x][Ctrl-b]        완충기목록을 현시한다
```

이때의 출력은 아주 유익하다(그림 5-18). 즉 그것은 emacs에 의해 조종되는 완충기들의 모든 디스크파일들의 경로이름과 그것들의 형태, 크기를 보여 준다.

```
MR  Buffer              Size   Mode          File
--- ------              ----   ----          ----
    dentry1.sh           733   Shell-script /home/romeo/project5/dentry1.sh
    convert.sh           520   Shell-script /home/romeo/project5/convert.sh
    *Completions*        172   Completion List
    conv2pm6.sh          322   Shell-scritpt /home/romeo/project5/conv2pm6.sh
    *scratch*              0   Lisp Interaction
*   *Messages*           240   Fundamental
*%  *Buffer List*        431   Buffer Menu

--1-:%*-F1  *Buffer List*   (Buffer Menu Abbrev)--L3--All-----------------------
```

그림 5-18. [Ctrl-x][Ctrl-b]에 의하여 현시되는 완충기목록

참고

만일 여러개의 파일들을 가지고 작업해야 한다면 emacs를 여러개의 파일이름과 함께 호출한다. emacs는 모든 파일들을 위한 완충기들을 만들며 그 완충기들은 완충기목록에 격납된다. [Ctrl-x][Ctrl-b]를 리용하여 그 목록을 현시할수 있으며 분리된 창문에서 편집하기 위한 임의의 완충기를 선택할수 있다. 또한 [Ctrl-x]b를 사용하여 현재의 창문을 다른 완충기로 교체할수 있다.

때때로 보존되지 않은 모든 변경을 무시할 필요가 있을수도 있다. 아래의 지령을 리용하면 그 파일의 마지막보관판본을 재적재할수 있다.

```
[Alt-x]revert-buffer
```

이것은 완충기내용을 디스크파일로 갱신한다. 이 지령은 파일의 자동보관된 판을 읽어 들이는 recover-file지령과 다르다.

참고

emacs에서 [Ctrl-x]b지령은 또 하나의 목적을 위하여 리용될수 있다. 그것은 어떤 파일이름과도 련관되지 않는《잡기록》완충기(scratch buffer)를 만드는데 사용될수 있다. 만일 emacs 작업의 도중에 누군가로부터 전화호출을 받는다면 주해를 만들기 위하여 이 잡기록완충기를 만든다.

5.15.3 파일과 지령출력의 삽입

유표의 현재위치에 어떤 파일의 내용을 삽입할수 있다. 즉 다른 파일을 편집한 다음 거기서부터 그 내용을 복사할 필요가 없으며 아래의 지령을 사용하면 된다.

```
[Ctrl-x]inote1        프롬프트에 입력된 파일이름
```

또한 파일안에 지령의 출력을 배치할수도 있다.

```
[Ctrl-u][Alt-!]date        파일에 있는 지령출력
```

이것은 자기들의 본문에서 지령출력을 서술할 필요가 있는 문서제작자들에게 아주 쓸모 있는 기능이다. 이 출력을 어떤 파일에 보관한 다음 그 파일을 읽어 들일 필요는 없다. 파일과 완충기조종지령들을 표 5-10에 보여 준다.

표 5-10. emacs에 있는 파일과 완충기조종지령들

지 령	기 능
[Ctrl-x][Ctrl-f]	현재의 파일편집을 정지하고 또 다른 파일을 편집
[Ctrl-x][Ctrl-v]	또 다른 파일에로 현재완충기를 치환
[Alt-x]revert-buffer	마지막으로 보존된 현재파일의 편집을 적재(Microsoft Windows에서 Revert와 같은)
[Alt-x]recover-file	자동보관된 파일을 적재
[Ctrl-x]b[Enter]	최근에 편집된 완충기에로 되돌림
[Ctrl-x]and select from list	또 다른 완충기의 편집
[Ctrl-x][Ctrl-b]	완충기목록표시
[Alt-x]shell	shell로 지적된 완충기에서 UNIX쉘대화를 시작
[Ctrl-x]i foo	현재유표위치에서 foo파일을 읽기
[Ctrl-u][Alt-!]head -3 foo	현재유표위치에 foo의 첫 3개 행을 읽기
[Ctrl-x]k	현재완충기를 없애기

5.16 쉘에로의 탈퇴

편집기를 완료하지 않고 쉘에로의 림시탈퇴를 할수 있다. 일반적으로 일감조종기능을 가지는 임의의 쉘에서 [Ctrl-z]는 응용프로그람을 중지하고 쉘프롬프트를 발생시킨다. 이 기능은 vi와 emacs에서 다 동작하지만 emacs는 자기자체의 별도의 지령을 가지고 있다. 다음의 조작을 눌러 보자.

```
[Ctrl-x][Ctrl-z]
[2]+ Stopped            emacs emfile
$ _
```

이때 편집이 중지되고 쉘프롬프트가 돌려 지는것을 보게 된다. 사용자는 쉘에서 몇가지 작업을 수행하고 fg를 입력하여 emacs로 돌아 갈수 있다. fg는 마지막전경일감을 재개하는 쉘의 내부지령이다.

만일 쉘이 일감조종을 지원하지 않는다면 그래도 편집창문에 쉘프롬프트를 가질수 있다. shell지령을 사용하여 보자.

```
[Alt-x]shell            vi에서 :sh와 같다
```

이것은 현재의 창문을 shell이라는 이름을 가진 새로운 완충기로 바꾼다. 프롬프트에서 진행한 작업은 이 완충기에 격납된다. 이것은 모든 건조작들과 지령출력이 이 완충기안에 보관된다는것을 의미한다. [Ctrl-x]b[Enter]를 리용하여 종전의 완충기로 즉시 돌아 갈수 있다. 프롬프트에서 exit는 그 쉘은 없애지만 창문을 없애지는 못한다.

만일 하나의 지령만을 실행하려고 한다면 쉘프롬프트에로 탈퇴할 필요가 없다.

```
[Alt-!]          실행될 UNIX지령을 문의한다
```

실행시킬 지령을 건으로 입력하면(!를 위하여 필요하다면 [Shift]건을 사용하여) emacs는 별도의 창문(*Shell Command Output*로 불리우는 완충기)에서 그 지령을 실행한다. 하지만 유표는 현재창문에

남아 있을것이다.

참고

만일 현재의 체계에서 script지령을 리용할수 없다면 shell이라는 이름을 가진 완충기를 보관하는것으로 쉘프롬프트에서 진행된 UNIX작업을 기록할수 있다. 이 완충기는 [Alt-x]shell을 리용하여 만든다.

5.17 도움말기능의 사용([Ctrl-h])

emacs는 지령들과 건반, 방식, 변수들에 대한 상세한 정보를 제공하는 아주 정교한 도움말기능을 가지고 있다. 더우기 훈련문서(tutorial)나 다중준위info문서들도 호출할수 있다. 이와 같이 그것들은 알려진 체계의 모든 면에서 사용자를 방조할수 있다. 우리는 이 장에 모든 기능들을 설명할수는 없으며 다만 매일 요구되는것들만을 론의할것이다. 도움말기능은 여러개의 모듈로 편성되며 [Ctrl-h]를 리용하여 그것들을 모두 불러 낸다(표 5-11).

표 5-11. emacs의 도움말기능지령

지 령	취급내용
[Ctrl-h]k	건누르기에 의하여 수행되는 기능(세부)
[Ctrl-h]c	건누르기에 의하여 수행되는 기능(한행)
[Ctrl-h]f	지령에 의하여 수행되는 기능
[Ctrl-h]w	지령에 쓰이는 건맺기
[Ctrl-h]v	변수의 기능과 그것의 현재설정
[Ctrl-h]a	개념을 사용하는 지령들
[Ctrl-h]t	훈련문서를 실행
[Ctrl-h]i	info읽기프로그람을 실행

5.17.1 건에 의한 도움말호출([Ctrl-h]k)

일반적으로 초학자들이 접하게 되는 두가지 문제점이 있다. 즉 그들은 건입력렬의 기능이나 지령의 기능을 기억할수 없다. 첫번째 문제는 [Ctrl-h]k(key)에 의하여 조종된다. 이 건을 누르면 emacs는 건렬을 입력할것을 요구한다.

```
Describe key: [Ctrl-y]          [Ctrl-y]를 위한 도움말찾기
```

이것은 복사 또는 삭제된 본문을 붙이는데 사용되는 지령이다. emacs는 즉시 분리된 창문에서 그 건입력렬의 상세한 서술로 응답한다.

```
C-y runs the command yank
   which is an interactive compiled Lisp function.
(yank &optional ARG)


Reinsert the last stretch of killed text.
More precisely, reinsert the stretch of killed text most recently
killed OR yanked. Put point at end, and set mark at beginning.
With just C-u as argument, same but put point at beginning (and mark at end).
```

```
With argument N, reinsert the Nth most recently killed stretch of killed text.
See also the command M-y.
```

이 설명은 매우 구체적이며 아주 쓸모 있다. 유표는 여전히 이전 창문에 남아 있기때문에 [Ctrl-x]1을 리용하여 도움말창문을 닫을수 있다.

5.17.2 기능이름에 의한 도움말호출([Ctrl-h]f와 [Ctrl-h]w)

emacs는 수천개의 지령들을 지원하며 그 모든 지령들의 기능을 기억한다는것은 사실상 불가능하다. 이 지령들의 일부는 건맺기를 가지며 일부는 가지지 않는다. [Ctrl-h]f와 [Ctrl-h]w는 지령들에 대하여 더 알려고 하는 경우에 사용하는 건입력렬이다. 지령이 무엇을 하는지 알기 위해서는 [Ctrl-h]f (function)를 사용하고 다음에 프롬프트에 그 지령이름을 입력한다.

```
Describe function (default *): recover-file
```

앞에서와 같이 emacs는 recover-file지령에 의해서 수행되는 기능을 서술한 별도의 창문을 연다.

```
recover-file is an interactive compiled Lisp function.
(recover-file FILE)
Visit file FILE, but get contents from its last auto-save file.
```

이 서술은 창문안에 맞출수 있도록 충분히 짧지만 흔히 본문의 나머지를 보기 위하여 다른 창문을 흘러 보내야 할 필요가 있을것이다([Ctrl][Alt]v를 리용하여). 반대로 흘러 보낼수는 없으며 그렇게 하려면 먼저 [Ctrl-x]o를 리용하여 그 창문에로 이동하여야 할것이다.

때때로 직면하게 되는 또 하나의 문제가 있다. 즉 건맺기가 없는 지령이름을 기억하는 경우이다. 실례로 독자는 아마 대화형치환이 질문치환지령에 의하여 수행된다는것을 기억할것이다. 하지만 지름건으로 동작하는 건맺기를 다시 호출할수 있는가? [Ctrl-h]w지령을 사용하여 보자(where-is).

```
Where is command (default *): query-replace
```

emacs는 소형완충기에 한행분의 통보를 보여 준다.

```
query-replace is on M-%, menu-bar search query-replace
```

이것은 지름건으로서 [Alt-%]를 보여 준다. 뿐만아니라 그것은 화면의 꼭대기에서 리용할수 있는 차림표선택항목 Search>query-replace에로 사용자를 안내한다. X Window체계에서 emacs를 사용할 때 이 차림표기능의 우점을 얻게 될것이다.

5.17.3 훈련문서 및 info문서의 보기([Ctrl-h]t와 [Ctrl-h]i)

마지막으로 아주 쓸모 있는 다른 2개의 도움말기능들을 보기로 하자. 사용자는 [Ctrl-h]t(tutorial)를 리용하여 emacs훈련문서를 호출할수 있다. 이것은 편집기에 대한 훌륭한 입문이며 초학자들을 위한 좋은 읽기자료를 봉사한다. 이전의 완충기에로 탈퇴하기 위하여 [Ctrl-x]b를 사용하자.

훈련문서보다 더 상세한것이 [Ctrl-h]i(info)로 호출되는 info문서이다. 이 문서는 현재 사용하고 있는 체계에 따라 "INFO tree"의 꼭대기에 갈수 있게 한다. Linux체계들은 많은 지령들에 대한 다량의 info문서를 제공하며 이 모든 지령들을 보여 주는 열린 화면을 볼수 있다. 이제 창문에서 유표를 이동하여 emacs를 보여 주고 있는 행을 찾아 보자. emacs문서의 열린 페지를 보기 위해서는 [Enter]건을 누른다.

사용자는 info문서가 여러개의 마디(준위)들로 편성되도록 다시 호출할수 있다(2.8). 마디들은 행의 시작에서 *로 표시된다. 임의의 마디들에 유표를 가져 갈수 있으며 그 제목에 대한 구체적인 정보를 얻으려면 [Enter]를 누른다. 이전 준위로 돌아 가기 위해서는 u를 사용한다. info방식에서 탈퇴하기 위해서는 q를 사용한다. info지령들에 대한 지식을 상기하기 위해서는 표 2-4를 찾아 보시오.

지금까지 설명된 emacs의 기능들은 그날그날의 편집과제를 수행하는데는 아주 충분할것이다. 이 기술을 사용하는것이 편리하다는것을 느낀 다음에는 작업능률을 개선할수 있는 몇가지 강력한 기능들을 더 사용하려고 할수도 있다. 만일 지금까지 emacs를 잘 알고 있다면 계속 읽어 나가시오.

5.18 본문에 표식달기

사용자는 파일에서 몇개의 위치들을 표식할수 있으며 후에 그것들을 찾아 낼수 있다. 서표(emacs에서도 제공된다.)와 달리 이 표식(mark)들은 눈에 보이며 편집기에서 탈퇴할 때 없어 진다. 표식지령과 위치찾기지령은 다 유일적인 표식을 만들기 위한 어떤 문자와 함께 사용되여야 한다. 어떤 위치를 문자 q로 표식하려면 필요한 위치에 유표를 이동하고 [Ctrl-x]/q를 누른다.

유표를 표식된 위치로부터 멀리 이동한후에 [Ctrl-x]jq를 사용하여 이 표식에로 되돌아 갈수 있다.

이러한 방법으로 서로 다른 자모문자들을 사용함으로써 파일안에서 26개까지의 위치를 표식할수 있다. 표식을 붙이기 위해서는 /을 사용하고 그 표식에로 이행하기 위해서는 j를 사용한다는것을 기억하시오.

서표의 리용

표식들은 편집기에서 탈퇴하자마자 곧 사라진다. 때때로 서로 다른 등록부들에 존재하는 여러개의 파일들을 편집한다면 틀림없이 파일을 접근하려고 할 때마다 그 파일의 경로이름을 입력해야 하는 권태감을 느낄것이다. emacs는 현재파일뿐아니라 임의의 위치에 있는 임의의 파일에서 위치를 표식하는 **서표**(bookmark)기능을 가지고 있다. 이 기능을 사용하면 다른 파일들을 최소의 건조작으로 호출할수 있다.

이름이 news.lst인 현재파일에서 현재위치에 서표를 붙이려 한다고 하자. 이때에는 다음의 지령을 사용한다.

```
[Ctrl-x]rm
```

프롬프트 Set bookmark (news.lst):가 소형완충기안에 나타난다. emacs는 표식의 이름을 입력할것을 요구하며 기정으로 현재의 파일이름을 제공한다. 서표의 이름은 꼭 단일문자여야 한다는 법은 없으며 이름안에 공백을 가질수도 있다.

일단 서표를 정의하였다면 다음의 지령을 사용하여 임의의 파일로부터 그 페지에 접근할수 있다.

```
[Ctrl-x]rb
```

참고 사용자가 정기적으로 편집하는 파일 특히는 .profile, .exrc, .emacs 등과 같은 체계의 구성파일을 위해서 서표를 설정할수 있다. emacs는 모든 서표들을 기억하며 홈등록부안에 있는 파일 .emacs.bmk에 그것들을 보관한다.

5.19 본문려과

vi에서와 같이 emacs도 UNIX지령을 실행하여 화면상의 본문을 수정할수 있게 한다. 이것은 어떤 지령으로부터 입력을 받으며 또 다른 지령에로 출력을 보내는 그러한 지령들을 리용할 때에만 가능하다.

이 지령들을 **려파기**(filter)라고 하며 이 책에서는 체계에서 리용할수 있는 모든 중요한 려파기들을 취급한다.

UNIX의 sort지령(려파기)은 파일의 내용을 정돈할수 있을뿐아니라 emacs화면의 내용도 정돈할수 있다. 일반적으로 본문을 려파하기 위해서 아래와 같이 하여야 한다.

1. 먼저 [Ctrl-x][Ctrl-x]를 사용하여 점의 위치를 확인하면서 구역을 선택하고 필요하다면 표식을 붙인다.
2. [Ctrl-u][Alt-\]을 사용한다. 이 건입력렬을 리용할 때 emacs는 다음의 지령을 요구한다.
   Shell command on region: sort
3. 실행해야 할 UNIX지령을 입력한다. sort를 입력하고 [Enter]를 누르면 동일한 창문에 려파된 출력이 나타난다.

vi를 리용하여 진행하였던 련습(4.19)을 반복하기 위해서는 아래의 건렬을 입력하여야 한다.

1. [Alt-x]goto-line[Enter]21: 21행으로 간다.
2. [Ctrl][Spacebar]: 본문의 시작에 표식을 준다.
3. [Alt-x]goto-line[Enter]40: 40행으로 간다.
4. [Ctrl-u][Alt-|]sort: 구역에서 sort지령을 실행한다.

주해

출력을 분리된 창문에 나타내려면 대체로는 같으나 다른것은 [Ctrl-u]를 [Alt-|]와 함께 사용하지 않는다.

참고

emacs는 행들을 정렬하기 위한 특수한 지령을 가지고 있다. 구역을 선택한 다음 [Alt-x]sort-lines를 사용하면 원래의 행들이 정렬된 출력으로 교체된다.

5.20 다중본문구역의 저장

vi와 마찬가지로 emacs는 몇개의 특수한 완충기들(영문자를 따서 이름 지은)에 26개까지의 본문블로크를 보존하는 기능을 지원한다. 이 완충기들을 사용하려면 첫째로 매 완충기를 위한 구역을 정의해야 한다. 일단 구역이 정의되면 아래의 지령들을 임의의 완충기이름(실례로 v)과 함께 사용한다.

```
[Ctrl-x]xv              본문이 v완충기에 복사된다
[Ctrl-u][Ctrl-x]xv      본문이 v완충기에로 삭제된다
```

그 구역안의 본문은 완충기 v에로 복사 또는 삭제된다. 새로운 위치로 이동하여 다음의 지령을 사용하여 보자.

```
[Ctrl-x]gv
```

우리는 vi를 리용하여 류사한 련습을 수행하였으며 따라서 간단한 실례를 가지고 우리의 지식을 공고화하여 보자. 우리는 어떤 파일로부터 또 하나의 파일 foo에 10개의 행을 복사하여 30행아래에 이 행들을 놓을것이다. 우리가 해야 할 일은 다음과 같다.

1. 10개의 행으로 구성된 구역을 선택한다.
2. [Ctrl-x]xa를 사용하여 그 행들을 완충기 a에 복사한다.

3. [Ctrl-x][Ctrl-f]foo를 리용하여 파일 foo에로 절환한다.
4. [Alt-x]goto-line[Enter]31을 사용하여 31행으로 이동한다.
5. [Enter]를 눌러 빈 행을 삽입한다(이때 유표는 32행에 위치한다).
6. [Ctrl-p]를 사용하여 31행으로 되돌아 간다.
7. [Ctrl-x]ga를 사용하여 30행아래에 그 10개의 행을 놓는다.

이름을 가진 완충기들의 이러한 기능을 리용하여 한개의 파일에서만이 아니라 한 파일로부터 다른 파일에로 본문을 자유롭게 복사하고 이동할수 있다. 그러나 언제든지 편집기에서 벗어 나지 말아야 한다는것을 기억하여야 한다.

5.21 다중삭제의 회복([Alt-y])

일정한 규칙을 따라 제공된 많은 삭제들을 회복할수 있다. emacs는 제거고리안에서 마지막 30개 항목의 회복을 허용한다. 그 항목들은 삭제된것들만이 아니며 복사된 본문도 제거고리로 간다. 그것들은 완전한 행이 되여야 할 필요가 없으며 제거조작에 의하여 삭제된 임의의 본문들이 회복될수 있다. 이것은 기본적으로 [Ctrl-d]가 아닌 임의의 지우기조작을 의미한다.

우리는 이미 [Ctrl-k]와 [Alt-d]를 리용하여 본문을 삭제하는것을 배웠다. 또한 [Ctrl-w]과 [Alt-w]를 각각 리용하여 구역안의 본문을 삭제하거나 복사하였다. 이 기술을 사용하여 여러번의 삭제 또는 복사를 진행하였을 때에는 새로운 위치에로 이동하여 [Ctrl-y]를 써서 가장 최근의 삭제나 복사를 회복한다. 만일 그것이 정확하지 않다면 이때에는 [Alt-y]를 사용한다.

`[Alt-y]` 초기의 [Ctrl-y]다음에만 동작한다

이것은 하나의 건조작으로 마지막회복 또는 복사를 취소하고 다음항목을 회복한다. 이 방법으로 제거고리로부터 30개까지의 항목들을 회복할수 있으며 그것은 대부분의 목적에 아주 적합할것이다.

5.22 본문의 생략(abbrev-mode)

emacs는 긴 단어들을 생략할수 있게 한다. 생략은 **전역적**(모든 방식에서 리용가능한)일수도 있고 **국부적**(현재방식에서만 유효한)일수도 있다. 우리는 이 방식들의 차이를 취급할 필요가 없으며 다만 전역방식에서 작업하고 있다고 가정한다. vi에서와 달리 사용자는 이 기능을 사용하기전에 생략방식(기정적으로는 생략방식이 아니다.)으로 절환하여야 한다. 그것을 절환하기 위해서는 다음의 지령을 사용한다.

```
[Alt-x]abbrev-mode[Enter]
```

그러면 방식행에 단어 Abbrev가 나타날것이다. 일단 그 방식이 가능해 지면 사용자는 본문을 입력하는 입력화면상에 략어(abbreviation)를 먼저 입력하는것으로 그 략어를 정의할수 있다. 실례로 Java 지령 System.out.println을 sopl로 생략하자. 먼저 임의의 행에 sopl을 입력하고 [Ctrl-x]를 누른 다음에 aig문자렬을 입력한다.

`sopl[Ctrl-x]aig` 국부생략을 위해서는 `ail`을 사용한다

이제 emacs는 확장어(expansion)를 요구한다. 소형완충기에 System.out.println를 입력하면 sopl

이 이 문자렬로 확장되는것을 보게 될것이다. 이것으로 생략을 정의하였다.

기동시에 모든 략어들이 필요하다고 결정하였다면 여기서부터 그것들을 보존하여야 한다. [Alt-x]write-abbrev-file을 사용하고 략어들을 격납할 파일의 이름(실례로 emacs-abbrevs)을 입력한다. 이것은 그 파일에 모든 략어들을 보관한다. 이제는 기동시에 생략을 가능하게 하고 이 파일을 적재하여야 한다는것을 emacs에게 통지하여야 한다. .emacs에 아래의 행들을 배치한다.

```
(setq-default abbrev-mode t)            t는 true값을 가리킨다
(read-abbrev-file "~/.emacs_abbrevs")
```

이것들은 LISP로 서술된 행들이며 그러므로 그것들을 해석하려고 지나치게 걱정할 필요는 없다. 이제는 기동시에 략어들을 리용할수 있다. setq-default는 전역변수들을 설정하기 위해서 사용되는 지령이며 우리는 그에 대하여 간단히 알아 볼것이다. 만일 추가적인 략어들이 그 파일에로 가도록 하려고 한다면 .emacs안에 추가적인 항목을 배치한다.

```
(setq save-abbrevs t)
```

사용자는 략어들을 렬거(list-abbrevs)하거나 편집(edit-abbrevs)할수 있으며 불가능(kill-all-abbrevs)하게 할수 있다. 이 모든 지령들의 앞에는 [Alt-x]가 배치되여야 한다.

참고

emacs는 략어를 완전하게 정의하지 않아도 되는 재치 있는 생략기능을 가지고 있다. 만일 이미 긴 단어로 타자하였다면 다음번에는 그것을 유일하게 하는데 충분한 단어를 입력한 다음 [Alt-/]를 누른다. emacs는 그 략어를 완성하여 준다. 만일 그것이 현재문서에서만 요구되는 략어이라면 그것을 정의하는 대신에 우와 같이 할수 있다.

5.23 건반의 전용화

emacs에서 누르는 모든 유효한 건입력렬들은 emacs의 내부지령에 속박된다. emacs는 수천개의 그러한 지령들을 가지고 있으며 그것들중 일부만이 실제로 그 건들에 속박된다. 자주 사용되는 건입력렬을 대응시키기 위해서는 global-set-key기능을 사용하여야 한다.

vi사용자(30행에 이동하기 위해서는 30G를 사용한다.)들은 일반적으로 특정한 행번호에로 이동하기 위하여 [Alt-x]goto-line을 사용하여야 한다는것을 불편스럽게 여길것이다. 우리는 이를 위한 건맺기를 가져야 하며 따라서 이 목적을 위하여 [Ctrl-x]w를 사용하기로 계획한다. 그러나 그전에 그 건이 이미 어떤 결합을 가지고 있는가를 검사하여 보자. [Ctrl-h]를 리용하여 도움말을 펼친 다음 건입력렬을 위한 프롬프트가 나타나도록 c를 누른다.

```
Describe key briefly: C-x w             [Ctrl-x]w를 입력한다
C-x w is undefined
```

이 건조합(key combination)이 정의되지 않았다는것을 알게 된 지금 global-set-key지령을 사용하여([Alt-x]와 함께) goto-line지령을 대응시키자. emacs는 2개의 입력을 제공할것을 요구한다.

```
Set key globally: C-x-w          건입력렬
Set key C-x w to command: goto-line            대응되는 지령
```

이제는 goto-line지령이 [Ctrl-x]w에 속박된다. 어떤 행에 이동하기 위해서는 이 건입력렬을 누르고 행번호를 입력하면 된다. 이 결합을 영구화하려면 아래의 항목을 .emacs파일의 끝에 배치한다.

```
(global-set-key "\C-xw" 'goto-line)              여기에 하나의 '가 있다
```

맺기가 겹인용부호로 둘러 막힐것을 요구하는 특이한 LISP문법을 주의하시오. 실행될 지령의 앞에는 반드시 한개의 외인용부호가 놓여야 한다. 사용된 규약을 주의하시오. 즉 \C-x는 [Ctrl-x]이다. 사용자들이 리용할수 있는 다른 확장문자렬(escape sequence)들도 존재한다. 즉 [Esc]를 위한 \e와 [Enter]를 위한 \r이다.

참고

emacs의 도움말기능을 사용하여 건입력렬이 내부적으로 실행되는 emacs지령들을 찾아 보자. [Ctrl-h]c(혹은 k)를 리용하여 도움말을 호출하고 임의의 입력렬을 입력한다. emacs는 실행될 내부지령의 이름을 알려 줄것이다. 만일 그것이 정의되지 않았다면 그것을 대응목적에 안전하게 사용할수 있다. 만일 그것이 정의되여 있다 해도 존재하고 있는 기능이 실제로 필요한가를 자체로 결정한다. 만일 그렇지 않으면 그것을 재정의할수 있다.

5.24 마크로의 리용

global-set-key기능은 emacs내부지령에 건입력렬을 대응시키는데 쓸모가 있다. 하지만 우리는 때때로 한 묶음의 건조작을 간단한 입력렬에 대응시킬 필요가 있다. vi에서는 :map지령이 이 일감을 조종하지만 emacs는 **마크로**(macro)를 사용할것을 요구한다. 마크로는 한 묶음의 건입력렬을 실행하는 사용자정의지령(user-defined command)이다.

마크로정의는 지령 [Ctrl-x](으로 시작한다. 일단 그 지령이 눌리우면 emacs는 기록방식으로 들어가며 사용자의 모든 건조작을 기록한다. 그 정의는 사용자가 [Ctrl-x])를 누를 때 끝난다(물론 이 조작은 기록되지 않는다).

emacs는 현재행을 복사하기 위하여 짧은 건입력렬를 가질것을 요구하며 따라서 우리는 마크로를 사용하게 된다.

```
[Ctrl-x](              정의를 시작한다
[Ctrl-a]               행의 시작에로 이동한다
[Ctrl][Spacebar]       표식을 설정한다
[Ctrl-e]               행의 끝으로 간다
[Alt-w]                구역을 복사한다
[Ctrl-x])              정의가 끝난다
```

이 마크로는 아직 이름을 가지고 있지 않지만 아래와 같은 방법으로 그것을 실행시켜 현재행을 복사할수 있다.

```
[Ctrl-x]e              그다음 [Ctrl-y]로 그것을 임의의 위치에 배치한다
```

이 입력렬은 call-last-kbd-macro지령을 실행한다. .emacs파일에 그것을 보관하기 위해서는 먼저 이름을 주는것이 필요하다(실례로 cpl). 이제 지령 [Alt-x]name을 호출하고 [Enter]건을 누른 다음 이 마크로에 이름 cpl을 입력하자.

```
Name for last kbd macro: cpl[Enter]
```

이것은 임의의 행으로 가서 [Alt-x]cpl지령을 주어 그 행을 복사할수 있다는것을 의미한다. 파일 .emacs를 열고 ([Ctrl-x][Ctrl-f]를 리용하여) 그 파일의 끝으로 이동한 다음 아래의 지령을 입력한다.

```
[Alt-x]insert-kbd-macro[Enter]          프롬프트에 cpl을 입력한다
```

emacs가 이 파일안에 삽입하는 LISP코드는 다음과 같다.

```
(fset 'cpl
   "\C-a\C-@\C-e\C-[w")          C-[w는 [Alt-w]이다
```

.emacs를 보관하면 cpl이라는 이름을 가진 이 마크로를 사용할수 있다. 사용자는 [Alt-x]와 마크로 이름을 사용하여 임의의 이름을 가진 마크로를 실행할수 있다.

emacs는 더 나아가서 사용자입력을 얻는것을 잠시 중지시키기 위한 점(point)들을 정의할수 있게 한다. 이 책에서는 그 기능에 대하여 론의하지 않을것이다.

5.25 편집환경의 전용화

emacs는 변수를 설정하기 위하여 두가지 형태의 지령을 사용한다. set-variable지령은 편집기안에서 변수들에 값을 할당하기 위하여 사용된다. 그러나 이 값들을 영구적인것으로 만들기 위해서는 .emacs안에서 setq와 setq-default지령들을 사용해야 한다. 이 변수들은 수값이나 문자렬 또는 론리값을 가질수 있다. 론리값들은 t 또는 nil일수 있다.

변수 auto-save-timeout를 고찰하여 보자. 이것은 비활성상태의 초단위시간을 의미하는 수값으로 설정되며 그 상태는 자동보관조작을 초래한다. 그 변수의 기정값을 알기 위하여서는 describe-variable 지령을 [Alt-x]와 함께 사용한다. 이것은 건맻기 [Ctrl-h]v를 가지는 도움말체계의 한 모듈이다. 이 건 입력렬을 입력한후에 emacs는 변수이름을 요구한다.

```
Describe variable: auto-save-timeout          완성하기기능을 사용한다
```

이제 emacs는 련관된 문서의 몇개 행을 가진 분리된 창문안에 현재의 설정을 보여 준다.

```
auto-save-timeout's value is
30


Documentation:
*Number of seconds idle time before auto-save.
Zero or nil means disable auto-saving due to idleness.
   .......
```

문서를 본 다음에는 [Ctrl-x]1을 리용하여 이 창문을 없앤다. set-variable지령을 리용하면([Alt-x]와 함께) 이 변수의 값을 60초로 변경시킬수 있다.

```
Set variable: auto-save-timeout
Set auto-save-timeout to value: 60
```

이것은 이 완충기를 사용할 때 작업의 휴식을 위한 값을 설정한다. 그것을 영구적인 값으로 만들기 위해서는.emacs에 setq나 setq-default를 사용해야 한다.

```
(setq auto-save-timeout 60)          닫긴 괄호에 주의할것
```

또한 자동보관조작을 초래하는 건조작의 회수도 설정할수 있다. 이것은 변수 auto-save-interval에

의하여 조종된다. 기정적으로 그것은 보통 300으로 설정되지만 .emacs에서는 그것을 변경시킬수 있다.

```
(setq auto-save-interval 200)
```

emacs는 **국부변수**(local variable)와 **전역변수**(global variable)사이에 차이를 만든다. 국부변수는 그것이 정의된 완충기에서만 적용되며 그 변수에 관하여 정의될수도 있는 임의의 전역값을 무시한다. setq지령은 국부지령을 설정하며 setq-default는 변수를 그의 기정값(전역값)으로 설정한다. 우리는 더 이상 상세한 내용을 취급하지 않을것이며 다만 setq지령이 동작하지 않는다면 그대신에 setq-default지령을 사용할것을 권고한다.

이제는 다른 중요한 일부 변수설정의 의미를 간단히 론의하자. 그 설정들이 .emacs안에 어떻게 배치되는가를 아래에 보여 준다.

```
(setq-default case-fold-search nil)
(setq tab-width 4)
(setq line-number-mode t)
(setq blink-matching-paren nil)
(setq kill-ring-max 50)
(setq abbrev-mode t)
```

case-fold-search가 nil로 설정될 때에는 탐색들이 대소문자를 구별하게 된다. 사용자는 타브문자가 차지할 공간의 개수를 설정할수 있다(tab-width). 만일 방식행에 행번호가 보이지 않는다면 line-number-mode의 값을 검사한다. blink-matching-paren이 t로 설정되면 유표는 { 혹은 (에로 이동하며 그다음 해당한 닫긴 괄호에로 돌아 온다(여기서는 그렇지 않다). 만일 제거고리안에 격납될수 있는 삭제의 기정개수가 적당치 않다고 확증되면 그 번호를 증가시킬수 있다(kill-ring-max). 때로는 생략기능을 해제할수도 있다(abbrev-mode nil).

참고

emacs 그자체를 리용하여 .emacs를 변경한 다음에는 그 변경을 유효하게 하기 위하여 emacs를 탈퇴 및 재개할 필요는 없다. [Alt-x]load-file지령을 사용하여 emacs가 .emacs를 다시 읽도록 하면 된다. vi에서는 이 기능을 리용할수 없다.

요 약

emacs는 방식 없는 전화면편집기이다. 마지막행(소형완충기)은 체계통보문들을 보여 주며 사용자입력을 받아 들인다. 그우의 행(방식행)은 편집기방식들과 파일이름을 현시한다. emacs는 기동시에 .emacs파일을 읽어 들인다.

emacs는 지령들을 실행하기 위하여 [Ctrl]과 [Meta]건을 사용한다. [Meta]건을 대신하여 [Esc]나 [Alt]건이 사용될수 있다. [Alt-x]와 그뒤에 지령본문을 입력하는 방법으로 emacs지령을 입력할수 있다. 지령렬은 [Ctrl-g]를 리용하여 취소할수 있다.

본문을 치환하기 위하여서는 overwrite-mode지령이 [Alt-x]와 함께 호출되여야 한다. 조종문자는 [Ctrl-q]의 뒤에 그 지령을 놓는 방법으로 입력된다.

emacs는 300개의 건조작이 수행될 때마다 파일을 자동보관한다. 사용자는 같은 이름을 리용하거나 ([Ctrl-x][Ctrl-s]) 다른 이름을 리용하여([Ctrl-x][Ctrl-w]) 파일을 보관할수 있다. 또한 보관하고 또는

보관함이 없이 편집기에서 탈퇴할수 있다([Ctrl-x][Ctrl-c]). 체계파괴가 일어 났을 때에는 자동보관된 파일을 회복할수 있다(recover-file).

지령을 반복하기 위하여서는 수자인수([Alt-n], 여기서 n은 옹근수)를 사용할수 있다. [Ctrl-u]가 지령앞에서 련속적으로 여러번 실행되면 그것은 반복회수를 4배로 증가시킨다.

[Ctrl]과 함께 사용된 b, f, p, n건들은 유표를 뒤로, 앞으로, 웃행과 다음행으로 이동시킨다. 사용자는 행의 시작([Ctrl-a])이나 끝([Ctrl-e])으로 이동할수 있다. [Alt]건이 b나 f와 함께 사용될 때에는 유표를 단어단위로 이동시킨다. 화면흘리기는 [Ctrl-v](전진방향)나 [Alt-v](후진방향)를 써서 진행한다.

특정한 행번호에로 이동하기 위해서는 goto-line지령을 요구한다. 또한 파일의 시작([Alt-<])이나 끝([Alt->])으로 직접 이행할수도 있다.

emacs는 먼저 시작을 표식한 다음 유표를 다른 끝으로 이동하는 방법으로 구역을 정의할수 있게 한다. 량끝점을 볼수 있으며([Ctrl-x][Ctrl-x]) 지어는 완충기전체를 구역으로 정의할수 있다([Ctrl-x]h).

문자를 삭제([Ctrl-d])할수도 있고 단어를 제거([Alt-d])하거나 행을 제거([Ctrl-k][Ctrl-k])할수도 있다. 현재유표의 위치로부터 행의 시작([Alt-0][Ctrl-k])까지 또는 행의 끝([Ctrl-k])까지 삭제할수 있다. 또한 빈행들의 묶음을 제거할수도 있다([Ctrl-x][Ctrl-o]). 삭제된 본문은 제거고리로 가며 후에 회복될수 있다.

구역안에서 본문을 삭제([Ctrl-w]) 또는 복사([Alt-w])할수 있다. 삭제 또는 복사된 본문은 [Ctrl-y]를 리용하여 새로운 위치에 배치한다.

단어들은 전반적으로([Alt-u]) 또는 첫 문자만([Alt-c]) 대문자로 된다. [Alt-l]은 단어들을 소문자로 변환한다. [Ctrl-x][Ctrl-u]를 리용하면 한 구역이 대문자로 변환된다. emacs는 본문의 일부를 입력하고 [Tab]건을 사용하면 그 본문의 나머지를 입력하지 않고도 그 지령을 완성하여 주는 지령완성하기기능을 제공한다.

편집은 동일한 건([Ctrl-x] 또는 [Ctrl-_])을 리용하여 취소하거나 재실행할수 있다. emacs는 취소가 완성되였을 때에는 재실행을 시작한다.

emacs는 증가탐색기능을 제공하며 여기서는 탐색문자렬의 일부가 입력되는 순간에 탐색이 시작된다. 그것은 또한 비증가탐색도 제공한다. [Ctrl-s]는 탐색의 시작과 반복을 위한 두 방법에서 다 리용된다.

emacs는 몇개의 특수한 문자들을 포함하는 표현을 리용하여 여러개의 문자렬을 나타낼수 있게 하는 정규식을 지원한다. [Ctrl][Alt]s를 리용하여 이 표현들을 두가지형의 탐색에 다 사용할수 있다. 비대화적으로(replace-string) 또는 질문치환기능을 사용하여([Alt-%]) 대화적으로 탐색과 치환을 진행할수 있다. 여기에도 정규식을 리용할수 있다(replace-regexp와 query-replace-regexp).

동일한 파일을 보기 위하여 화면을 2개로 가를수 있다([Ctrl-x]2). 또한 새로운 창문을 열수도 있으며([Ctrl-x]b) 창문들사이를 절환하거나([Ctrl-x]o) 창문크기를 증가시킬수 있다([Ctrl-x]^). 현재의 창문만을 현시하고 다른 모든 창문들을 제거할수 있다([Ctrl-x]1).

다른 파일의 내용을 삽입하거나([Ctrl-x]i) 다른 파일에로 이동할수 있으며([Ctrl-x][Ctrl-f]) 마지막으로 편집된 파일에로 돌아 가거나([Ctrl-x]b) 마지막으로 보관된 판본을 되살릴수 있다(revert-buffer).

[Ctrl-u][Alt-!]를 리용하여 편집기내부로부터 UNIX지령을 실행할수 있다.

emacs는 [Ctrl-h]를 공통적인 건조작으로 사용하는 폭 넓은 도움말기능을 지원한다. 건이 무엇을 하는지([Ctrl-h]k), 지령의 기능이 무엇인지([Ctrl-h]f), 건맺기가 무엇인지([Ctrl-h]w)를 찾아 낼수 있다. 또한 훈련문서([Ctrl-h]t)나 info문서([Ctrl-h]i)도 볼수 있다.

[Ctrl-x]/와 한개의 문자를 리용하여 본문안에 26개까지의 구역들을 표식할수 있으며 [Ctrl-x]j와 그 문자를 사용하여 표식된 위치에 접근할수 있다. 지어 어떤 위치에 서표를 붙일수 있으며([Ctrl-x]rm) 임의의 파일로부터 그것에 접근할수 있다([Ctrl-x]rb). 모든 서표들은 자동적으로 .emacs.bmk에 보관된다.

구역안에서 그 구역의 내용들을 려과하는 UNIX지령을 실행할수 있다([Ctrl-u][Alt-l]).

emacs는 26개까지의 완충기(a부터 z까지)를 사용하며 [Ctrl-x]x뒤에 그 완충기 이름을 사용하여 본문을 저장한다. [Ctrl-x]g와 그 이름을 리용하여 본문을 다른 파일에 배치할수 있다.

[Alt-y]를 리용하여 삭제된 본문을 회복할수 있으며 그 건이 눌리울 때마다 제거고리로부터 다음항목이 회복된다.

문자렬이 생략([Ctrl-x]aig)될수 있으려면 생략을 가능상태로 만들어야 한다(abbrev-mode). 임의의 emacs지령이 건입력렬에 속박될수 있다(global-set-key). 략어는 별개의 파일에 씌여 질수 있지만 건맺기는 .emacs에 저장된다.

마크로정의들은 [Ctrl-x](로 시작되며 [Ctrl-x])로 끝난다. 마지막으로 정의된 마크로를 실행할수 있으며([Ctrl]e) 마크로에 이름을 붙이거나([Alt-x]name) 그것을 .emacs에 저장할수 있다(insert-kbd-macro).

setq와 setq-default를 리용하여 변수들을 설정한다. 탐색이 대소문자를 구별하도록 할수 있으며(case-fold-seareh) 자동보관주기를 설정하거나 (auto-save-timeout와 auto-save-interval) 제거고리의 크기를 바꿀수 있다(kill-ring-max).

시험문제

1. 만일 방식행에 3개의 이음표(---)가 보인다면 그것은 무엇을 가리키는가?
2. 사용자가 틀린 지령을 입력하였는데 emacs가 입력을 더 요구하는것을 발견하였다. 이 혼란을 어떻게 수습하는가?
3. [Ctrl-h]가 왜 후진건으로 동작하지 않는가?
4. 만일 [Insert]건이 덧쓰기방식을 지원하지 않는다면 무슨 지령을 사용해야 하는가?
5. 사용자가 파일 foo1을 편집하고 있으며 이제 현재의 완충기를 foo2에 보관하려고 한다고 가정하자. 그것을 어떻게 할것인가?
6. 유표의 현재위치로부터 행끝까지의 모든 본문을 어떻게 지우는가?
7. 행을 지우지 않고 행으로부터 모든 본문을 어떻게 제거하는가?
8. 16개의 단어들을 삭제하는 가장 좋은 방법은 무엇인가?
9. 행의 40번째 문자를 어떻게 찾아 내겠는가?
10. 어느 지령을 가지고 화면을 흐르게 하는가?
11. 파일의 끝에 어떻게 가닿는가?
12. 행의 끝에 문자렬 */을 어떻게 추가하는가?
13. 단어 Richard Stallman을 어떻게 탐색하며 Richard를 Dick로 어떻게 바꾸는가?
14. 파일의 마지막보관판본을 어떻게 되살리는가?
15. 최소의 건조작을 사용하여 어떻게 CREATE를 DROP로 전역적으로 치환하는가?
16. 작업도중에 시간을 알려고 한다. emacs에서 탈퇴하지 않고 그것을 어떻게 알아 내는가?

련습문제

1. emacs가 기동시에 읽는 파일은 무엇인가? 그 파일을 읽지 않도록 하려면 어떻게 할수 있는가?
2. 문자렬 ^[가 보인다면 그것은 무엇을 나타내는가? 그것을 어떻게 입력하는가?

3. [Ctrl-d]와 [Alt-d]사이의 차이점은 무엇인가?
4. 수자인수와 만능인수사이의 차이점은 무엇인가? 그것들은 어느 건들을 사용하는가?
5. 수자인수를 사용하지 않고 20개의 행을 어떻게 지우겠는가?
6. [Alt-3][Ctrl-k]를 리용하여 3개의 행을 지우려고 시도하였지만 동작하지 않는다. 왜 그렇게 되였으며 어떻게 해야 하는가?
7. 만일 [Ctrl-k]를 4번 사용하여 본문의 4개 행을 지웠다면 어떻게 그 행들을 모두 회복할수 있는가?
8. 단어 Computer를 Comptuer로 틀리게 입력하였다. 그것을 어떻게 바로잡을것인가?
9. 어떤 파일안에 있는 1000개의 행을 모두 련결하여 한개의 행을 구성하려면 어느 편집기를 사용하여야 하며 어떻게 할것인가?
10. 현재의 작업에서 goto-line지령을 실행하였고 이제 그것을 다시 사용하려고 한다. 될수록 적은 건조작을 사용하여 그것을 수행하는 두가지 방법을 설명하시오.
11. 파일의 시작과 끝사이에서 어떻게 왔다갔다할수 있는가?
12. 지령완성하기기능을 리용하여 최소의 건조작으로 다음의 지령들을 완성하시오.

    (1) replace-string (2) recover-file (3) query-replace

13. 방금 취소한것을 어떻게 재실행하는가?
14. 파일의 전체 내용을 어떻게 소문자로 변환할것인가?
15. 작업을 취소하고 있을 때 완충기의 상태가 마지막보관판본과 같은 시점을 어떻게 결정하는가?
16. 다음의 지령들 가운데서 어느 지령이 취소될수 있는가를 설명하시오.

    (1) [Ctrl-f] (2) [Ctrl-p] (3) [Ctrl-d] (4) [Ctrl-t]

17. 매 문장의 끝으로 어떻게 이동하는가?
18. 처음 몇개의 치환을 대화식으로 수행하고 그 나머지를 자동적으로 수행하는것이 가능한가?
19. [Alt-x]query-replace를 리용하여 몇개의 치환을 진행하였다. 그 지령을 다시 입력하지 않고 어떻게 반복할것인가?
20. 현재창문에서 행의 개수를 4만큼 늘이려면 어떻게 하여야 하는가?
21. 현재행에서부터 파일의 시작까지 본문을 지우려면 어떻게 해야 하는가?
22. emacs에서 탈퇴하지 않고 어떻게 두번째 파일을 편집할수 있으며 그때 이 두 파일사이를 어떻게 절환할수 있는가?
23. 조종건들중 하나를 재정의하려고 한다. 그것이 이미 건맺기를 가지고 있는지 아닌지를 어떻게 알아내는가?
24. 완충기전체를 정돈하는 두가지 방법을 설명하시오.
25. 11행부터 30행까지를 어떻게 지우며 11행부터 20행까지 그리고 21행부터 30행까지를 어떻게 서로 다른 위치에 회복하는가? 주의하여야 할것은 무엇인가?
26. 한 파일로부터 다른 파일로 본문의 몇개 부분을 어떻게 복사하는가?
27. regular expression으로 확장될 략어 re를 정의하였고 re-rating을 입력하려고 할 때 그 확장어가 얻어 지는것을 발견하였다. 무엇을 할것인가?
28. 정의하지 않고 략어를 사용할수 있는가?
29. 파일에서 모든 꼬리부공백을 제거하는 마크로를 서술하시오.
30. emacs가 항상 마지막 50개의 삭제조작을 회복할수 있도록 하려면 어떻게 emacs를 전용화하는가?
31. recover-file과 revert-buffer사이의 차이점은 무엇인가. 일반적으로 어느 파일이 더 최근의것인가?

제 6 장. 파일체계

UNIX는 모든것을 파일로 고찰한다. 수만개의 파일들로 이루어 진 UNIX파일체계도 있다. 한개의 프로그람을 작성해 넣으면 체계에 한개의 파일이 더 첨부된다. 프로그람을 콤파일하고 실행하면 일부 파일이 더 첨부된다. 파일들이 늘어 나는데 맞게 그것들을 완전히 조직화하지 않으면 파일들을 호출하기가 실지로 불가능하다. 파일조직을 바로 하자면 등록부에 기초한 정교한 파일체계(사용자가 그 등록부에 《들어 갈수》도 있고 등록부들사이에 파일을 옮길수 있는)가 요구된다.

파일체계는 UNIX에서 간단하고 개념적으로 명백한 특징들중의 하나이다. 이 체계가 쓸모 있다는것이 확증되였기때문에 Windows를 포함하고 있는 다른 체계들에 의해 광범히 리용되고 있다. 이 장에서는 파일체계에 있는 등록부들을 취급하며 등록부들안에서 파일들의 복사와 제거 그리고 파일표시와 인쇄하는 방법을 배우게 된다. 또한 UNIX가 제공하고 있는 디스크공간을 효과적으로 리용하기 위한 지령들을 취급한다. 이 장은 파일과 등록부들을 취급하며 그 속성은 취급하지 않는다. 파일속성들은 지령들과 함께 다음장에서 취급하게 된다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- UNIX가 파일들을 세 부류로 나누는 방법을 배운다(6.1).
- 파일이름을 구성하는데서 고려할 점을 파악한다(6.2).
- 파일들사이에서 어미-새끼관계를 파악한다(6.3).
- UNIX파일체계에 있는 중요한 등록부들에 대하여 배운다(6.4).
- 절대경로이름과 상대경로이름사이의 차이점을 배운다(6.6, 6.8).
- pwd와 cd로 등록부를 검사하고 변경한다(6.5, 6.7).
- mkdir와 rmdir로 등록부를 만들고 제거한다(6.9, 6.10).
- cp, rm과 mv로 파일을 복사하고 제거하며 그 이름을 변경시킨다(6.11, 6.12, 6.13).
- cat로 파일을 현시하고 만든다(6.14).
- file로 파일의 형을 알아 낸다(6.15).
- lp(혹은 lpr)로 파일을 인쇄한다(6.16).
- df와 du로 디스크사용공간과 남은 공간을 알아 낸다(6.17, 6.18).
- compress, gzip와 zip로 파일을 압축한다(6.19).

6.1 파일

UNIX파일은 정보의 저장고로서 대부분이 문자렬이다. UNIX는 파일구조에서 제한을 주지 않는다. 파일은 사실상 사용자가 넣어 준 바이트들이며 그것은 원천프로그람, 실행가능한 코드 또는 그밖의 어떤것으로 될수 있다. 파일은 파일의 끝표식을 비롯해서 자기자체의 크기도 속성도 포함하지 않으며 지어 자기의 이름자체도 포함하지 않는다.

파일정의에는 등록부들과 장치들이 포함된다. 그것이 C프로그람이든 프로그람을 묶는데 사용하는 등

록부이든 그것들은 모두 UNIX개념에 있는 파일들이다. 또한 하드디스크, 인쇄기, 테프, CD-ROM구동기 또는 말단과 같은 모든 장치들도 파일이다. 쉘도 파일이며 핵심부도 파일이다. 그리고 UNIX는 사용자의 체계에 있는 주기억기를 역시 파일로 취급한다.

UNIX파일체계에서 가장 주목할만한 특징은 여러가지 형의 파일들사이에 차이가 적은것이다. 많은 지령들이 모든 형태의 파일들과 함께 동작하며 사용하고 있는 파일형태를 구체적으로 지적할것을 요구하지 않는다. 디스크파일에 접근하기 위해서 사용되는 지령들의 대부분은 테프구동기에도 접근한다. UNIX는 CD-ROM과 테프들에 접근하는 지령들을 따로 제공하지 않는다. 즉 같은 지령으로서 그것들에 접근한다.

UNIX는 모든것을 파일로 취급하며 파일은 대체로 세 부류로 나누어 진다.

- 보통파일(ordinary file): 정규파일(regular file)이라고도 한다. 문자흐름으로서의 자료만을 포함한다.
- 등록부파일: 다른 파일들과 등록부(directory)들의 이름이 들어 있는 폴더(folder)
- 장치파일: 모든 하드웨어장치들을 나타낸다.

파일을 세 부류로 나누는 리유는 파일의 속성이 파일형에 의존하기때문이다. 보통파일의 읽기허가는 등록부파일이나 장치파일의 읽기허가와 매우 차이난다. 게다가 등록부파일에 어떤것은 직접 넣을수 없으며 장치파일은 사실상 문자들의 흐름이 아니다. 대부분의 지령들이 모든 파일형태와 함께 동작하지만 일부 지령은 함께 동작하지 않는다. 파일체계를 옳게 리해하기 위해서는 이 파일들의 의미를 리해해야 한다.

주해

UNIX파일은 파일의 속성뿐아니라 파일의 끝표식도 포함하지 않는다. 파일이름도 역시 파일에 보관되지 않는다.

6.1.1 보통파일

전통적인 파일은 보통파일이다. 그것은 어떤 영구자석매체에 들어 있는 자료흐름으로 구성된다. 보통파일은 모든 자료, 원천프로그람, 목적코드, 실행가능한 코드, 모든 UNIX지령들, 사용자에 의해 만들어진 파일과 같은것을 포함한다. 앞으로 cat, ls와 같은 지령들은 보통파일 또는 정규파일로 취급된다.

보통파일의 가장 일반적인 형태는 **본문파일**(text file)인데 이 파일은 인쇄가능한 문자들을 포함하고 있는 정규파일이다. 우리가 일반적으로 쓰기하는 프로그람은 본문파일이며 UNIX지령들이나 혹은 C프로그람은 본문파일이 아니다. 본문파일의 특징적인 구조는 자료들을 이루는 매개 행이 행바꾸기(LF:linefeed)문자(ASCII 10진값 10)로 끝난다는것이다. UNIX체계에서 행바꾸기는 영어로 newline이라고도 하며 행바꾸기에 대해 말할 때 이것은 사실상 LF를 의미한다. 이 문자는 보이지 않으며 Enter건을 누를 때 체계에 의하여 발생된다.

6.1.2 등록부파일

등록부파일은 외부자료를 포함하지 않지만 파일의 일부 세부항목과 그것을 포함하고 있는 보조등록부들을 보존한다. UNIX파일체계는 이러한 등록부들과 보조등록부들로써 편성되며 필요할 때 만들수 있다. 때때로 특수한 응용프로그람에 속한 파일들의 모임을 묶어서 등록부안에 배치하는것이 필요하다. 같은 이름을 가진 파일들이 서로 다른 등록부에 배치될수 있다.

등록부파일은 파일의 이름과 식별번호(identification number)를 위한 두개의 마당을 포함한다(매개파일은 **색인마디번호**(inode number)라고 부르는 수를 가지고 있다). 만일 하나의 등록부가 10개 파일을 가지고 있다면 등록부파일에 10으로 기입될것이다. 그러나 등록부파일에 직접 쓰기할수는 없다. 보통파일이 만들어 지거나 제거될 때 대응하는 등록부파일은 파일과 관련된 정보들과 함께 핵심부에 의하여 자동적으로 갱신된다. 등록부파일에 대해서는 다음장에서 더 론의한다.

주해

등록부파일은 등록부안에 있는 모든 파일이름들을 포함한다.

6.1.3 장치파일

UNIX에서는 물리적인 장치들도 파일로 취급하기 위하여 파일에 대한 정의를 확대하였다. 이 정의는 인쇄기, 테프, 플로피구동기, CD-ROM, 하드디스크와 말단들을 포함한다. 처음에는 좀 당황할수 있지만 실지로는 유리하다. 디스크파일접근에 리용되는 지령들의 일부는 장치파일들에도 작용한다.

장치파일은 어떠한 자료도 포함하지 않는 특수한 파일이다(이 파일은 《문자렬》이 아니다). 장치를 지적한 지령의 출력은 파일이름과 련관된 물리적인 장치에 반영될것이다. 파일을 인쇄하기 위해 지령을 줄 때 실지로 인쇄기와 련관된 파일에 파일의 출력을 지정한다. 테프우에 파일을 여벌복사할 때 《상징적으로》 테프구동기와 련관된 파일을 사용하고 있다. 핵심부는 매개 장치들에 이러한 특수한 파일들을 배치하는것을 관리한다.

이제는 이 책에서 파일의 세가지 형을 리해하는데서 제기되는것이 없을것이다. 이 세가지에 대해서 일반적으로 파일이라는 용어로 표현하는 경우도 있지만 대체로는 보통파일을 의미한다. 그 용어의 실지 의미는 그의 문맥을 통하여 알수 있을것이다.

6.2 파일이름구성법

UNIX체계에서 대부분의 파일이름은 255개 문자로 이루어 진다. 만일 파일이름을 렬거할 때 255개이상 기입하면 첫 255개 문자만 체계가 인식한다. 그러나 일부체계는 오유통보문을 통지한다.

파일들은 확장자를 가질수도 있고 가지지 않을수도 있으며 실지로 /을 제외한 어떠한 ASCII문자로 이루어 질수 있다. 파일이름이 점으로 시작될수 있는것처럼 파일이름의 마지막에도 점이 놓일수 있다. 다음의 파일이름들은 유효하다.

```
index     .last_time     LIST.
```

만일 원한다면 파일이름에 조종문자 또는 인쇄할수 없는 다른 문자들을 사용할수 있다. 아래의 파일이름들은 류별나지만 유효하다.

```
^V^B^D-++bcd      -{}[]        @#$% *abcd             *앞에 공백
```

첫번째 파일이름은 3개의 조종문자들을 포함한다([ctrl-v]가 첫번째이다). 두번째 파일이름은 이음표로 시작된다. 세번째것은 이름에서 빈 공백을 명백히 보여 준다. 뒤에서 파일이름이 실지로 공백이나 인쇄할수 없는 문자를 포함하는가 안하는가를 알아 내는 방법을 서술한다.

많은 문자들이 지령행에서 사용될 때 특수한 의미를 가진다. 특히 쉘은 $, -, ?, *와 같은 문자들을 서로 다르게 취급한다. 지령들은 파일이름이 이 문자들을 가질 때 예측할수 없게 행동하므로 파일이름에

조종문자를 사용하는것을 피해야 한다. 사실상 파일이름을 조작할 때 다음과 같은 문자들만 사용하여야 한다.

- 자모와 수자
- 끝점(.)
- 이음표(-)
- 밑선(_)

UNIX는 확장자에서 제한이 없다. 쉘스크립트는 식별에서 도움이 될지라도 .sh확장자를 가질것을 요구하지 않는다. 모든 응용프로그람들은 확장자에서 제한이 있다. 례를 들면 C콤파일러는 .c확장자를 가지는 C프로그람을 요구하며 Java는 .java확장자를 가진 원천파일을 요구한다. Windows사용자들은 또한 다음과 같은 두가지 요점들을 명심해야 한다.

- 파일은 자기 이름에 많은 점들을 가질수 있다. 즉 a.b.c.d.e는 완전히 유효한 파일이름이다. 파일이름은 또한 점으로 시작되거나 그 점으로 끝날수 있다.
- UNIX는 다음의 경우를 갈라 본다. 즉 chap01, Chap01과 CHAP01은 서로 다른 3개의 파일이름으로서 같은 등록부안에 동시에 존재할수 있다.

참고

파일이름구성에서 자모, 수자, 점, 이음표 그리고 밑선외에 다른 문자를 사용하는것을 피하고 대문자는 적게 사용하여야 한다.

주의

파일이름의 시작에서 이음표를 절대로 사용하지 말아야 한다. 그것을 제거하는데 시간을 소비할것이다. 더우기 인수로 이 파일이름을 사용하고 있는 많은 지령들은 그것을 한개 선택항목으로 취급하며 오유를 통지한다. 례를 들어 파일이름이 -z라면 cat -z는 파일을 표시하지 않으며 무효선택항목으로 해석한다.

6.3 어미-새끼관계

UNIX의 모든 파일들은 다른것과 《련관》되여 있다. UNIX에서 파일체계는 계층적인 구조(뒤집어 놓은 나무)로 이루어 진 련관된 모든 파일들(보통파일, 등록부파일, 장치파일)의 모임이다. 이 체계는 Windows에서 채용되였는데 그림 6-1에서 시각적으로 보여 주고 있다.

UNIX파일체계에서 주목할만한 특징은 모든 파일에 기준점으로 되는 최상위(등록부)가 존재하는것이다. 이것을 **뿌리**(root)라고 하며 /으로 표시한다. 뿌리는 사실상 등록부파일이며 자기밑에 체계의 모든 보조등록부들을 가지고 있다. 이 보조등록부들은 자기밑에 보조등록부들과 다른 파일들을 더 가지고 있다. 례를 들어 bin과 usr는 뿌리등록부밑에 있는 등록부이며 두번째 bin과 lib는 usr등록부에 속해 있는 보조등록부들이다.

뿌리등록부와 다른 모든 파일은 어미를 가져야 하며 뿌리등록부에서 파일관계를 찾아 내는것이 가능해야 한다. 만일 파일의 제일 웃준위(ancestry)등록부를 뿌리등록부에서 추적할수 없다면 그 파일은 이 파일체계에 속한 파일이 아니다. 이것은 우리들의 가정에서처럼 조부모-부모-자식관계와 류사하게 생각하면 리해하기 쉽다. 즉 home등록부는 50romeo의 어미등록부이지만 뿌리등록부는 home의 어미등록부이며 romeo의 어미웃준위(grandparent)등록부이다. 만일 romeo등록부밑에 login.sql파일을 만들면 romeo는 이 파일의 어미등록부로 될것이다. 또한 어미-새끼관계에서 어미가 항상 등록부라는것이 명백

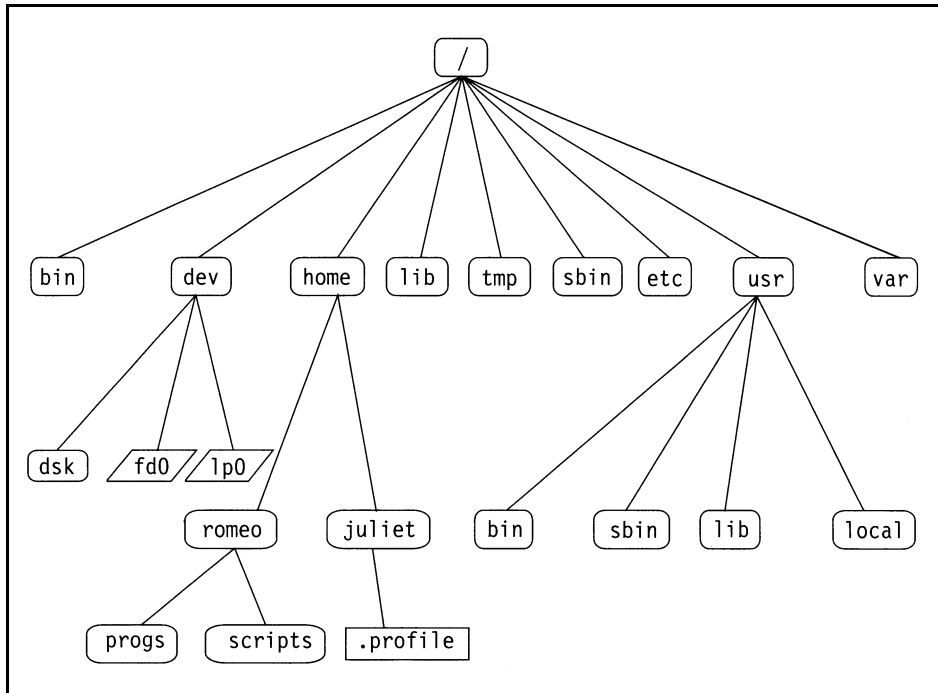



그림 6-1. UNIX파일체계

하다.

home과 romeo는 둘 다 적어도 한개 파일 혹은 한개 등록부의 어미이므로 등록부들이다. login.sql은 보통파일이며 자기밑에 등록부를 가질수 없다.

6.4 UNIX파일체계

UNIX파일체계의 구조를 보자. 이 구조는 끊임없이 변화되고 있다. 그림 6-1은 표준UNIX파일체계의 구조를 보여 준다. 실제로 뿌리등록부는 자기밑에 그림 6-1에서 보여준 것보다 더 많은 등록부들을 가지고 있지만 초기의 리해를 위해서 다음과 같은것을 알아야 한다.

- /bin과 /usr/bin: UNIX지령들이 들어 있는 등록부들이다(2진자료이므로 bin이라고 한다).PATH변수(2.2)는 이 등록부들을 목록으로 보여 준다.
- /sbin과 /usr/sbin: 사용자가 할수 없고 체계관리자만이 실행할수 있는 지령은 이 등록부가운데서 어느 한 등록부에 있을것이다. 낡은 체계에서는 이 등록부들이 존재하지 않았으며 /etc가 모든 관리파일들을 포함하고 있다. 이 등록부들에 있는 대부분의 지령들은 실행되지 않는다. 오직 체계관리자의 PATH만이 자기 목록안에 이 등록부들을 가지고 있다.
- /etc: 이 등록부는 체계의 구성파일들을 포함하고 있다. 이 등록부에서 본문파일을 편집하는데 따라 매우 중요한 체계기능을 변화시킬수 있다. /sbin과 /usr/sbin등록부가 없는 체계들은 /etc에 관리지령들을 가지고 있다.
- /dev: 이 등록부는 모든 장치파일들을 포함하고 있다. 이 파일들은 디스크공간을 차지하지 못한다. 이 등록부에 dsk와 rdsk와 같은 보조등록부들이 더 있을수 있다.
- /home: 모든 사용자들이 여기에 집결되여 있다. romeo는 /home/romeo에 자기의 home등록부

를 가지고 있다. 이전의 SVR4 체계들은 /usr에 home등록부를, 때때로 다른 등록부에 home등록부를 가진다. 체계관리자는 때때로 사용자들을 위해서 /usr2, /usr 등과 같이 서로 다른 등록부를 사용한다.

- /tmp: 림시파일들을 만들기 위한 등록부이다. 이 파일들은 체계에 의해서 정규적으로 만들어 진다.
- /var: 파일체계의 변수들이 들어 있는 등록부이다. 모든 인쇄일감들과 전자우편대기행렬 그리고 들어 오는 전자우편을 포함한다.
- /lib: 모든 서고파일들을 포함한다.

이밖에도 등록부들이 더 있는데 UNIX변종들마다 체계등록부들의 이름과 위치가 차이난다.

6.5 현재등록부알아보기(pwd)

UNIX의 주목할만한 특징은 파일과 마찬가지로 사용자도 등록가입시에 파일체계의 지정된 등록부에 배치된다는것이다. 사용자는 한개 등록부로부터 또 다른 등록부에로 이동할수 있으며 때로는 한개 등록부에만 배치된다. 이러한 등록부를 사용자의 **현재등록부**(current directory)라고 한다. 사용자는 현재등록부가 어느 등록부인가를 아는것이 때때로 필요하다. pwd지령으로 현재등록부를 알수 있다.

```
$ pwd
/home/romeo                이것은 경로이름이다
```

우에서 본것이 **경로이름**(pathname)이다. 등록부이름은 /으로 분리되여 있다. 이 경로이름은 뿌리등록부와 함께 현재등록부이름을 보여 준다. pwd지령은 여기서 현재등록부가 romeo등록부이며 home등록부가 romeo의 어미등록부라는것을 /으로 분리하여 보여 주고 있다.

홈등록부

체계에 가입할 때 UNIX는 사용자를 자동적으로 **홈등록부**(home directory) 혹은 **가입등록부**(login directory)라고 불리우는 등록부에 배치한다. 홈등록부는 등록자리를 열 때 체계에 의해서 만들어 진다. 만일 사용자가 romeo라는 이름으로 가입한다면 그는 경로이름 /home/romeo 혹은 /usr/romeo를 가지는 등록부에 도착할것이다.

쉘변수 HOME은 홈등록부를 알려 준다. 그것을 평가하기 위하여 쉘프롬프트에 다음의 지령을 입력한다.

```
$ echo $HOME
/home/romeo
```

홈등록부는 사용자등록자리가 열리는 때에 체계관리자에 의해서 결정된다. 이 경로이름은 /etc/passwd파일안에 보관된다. 이전의 체계들에서 /usr/romeo가 romeo의 기정홈등록부로 되지만 현재의 체계에서는 /home/romeo가 홈등록부로 된다. 또한 이전의 체계들에서는 /usr2/romeo혹은 /u/romeo와 같은 홈등록부들도 볼수 있다.

주해

홈등록부에 있는 foo파일을 지적하기 위해서는 $HOME /foo 혹은 ~/foo로 하는것이 보통이다. 이름짓기규정은 이 책에서 주었다.

6.6 절대경로이름

많은 UNIX지령들은 인수들로서 파일과 등록부이름들을 사용한다. 이 파일들과 등록부들은 현재등록부에 존재해야 한다. 실례를 들어 login.sql파일이 현재등록부에 존재할 때만

```
cat login.sql
```

이 동작할것이다. 그러나 만일 /home/romeo등록부안에 있는 login.sql파일을 /var등록부에서 호출하려고 한다면 명백히 우의 지령을 사용할수 없다. 현재등록부 /var로부터 login.sql파일을 호출하는 방법에는 두가지가 있다.

- 뿌리등록부를 파일에 대한 최종참조로 사용하는 절대경로이름으로 호출하는 방법. 여기서 모든 경로는 뿌리등록부로부터 발생된다.
- 현재등록부를 참조점으로 사용하고 그와 관련되는 경로만을 지정하는 상대경로이름으로 호출하는 방법

여기서는 간단히 상대경로이름을 고찰한다. login.sql에 대한 절대경로이름을 사용하자면 이 지령행을 /var로부터 사용하여 그 파일을 현시해야 한다.

```
cat /home/romeo/login.sql                어떠한 등록부로부터 사용될수 있다
```

여기서 cat는 절대경로이름을 사용하며 login.sql의 위치는 뿌리등록부(첫 사선)를 참고해서 렬거된다. 경로이름에 한개이상의 /이 있을 때 매개 /은 파일체계에서 한 준위아래로 내려 가야 한다. 즉 romeo는 home등록부밑의 한개 준위아래에 있으며 뿌리등록부보다는 2개 준위아래에 있다.

서로 다른 준위들을 경계 짓기 위해서 /을 사용하며 파일을 렬거할 때 파일을 유일하게 분간하는 기구(조종장치)를 가진다. UNIX체계에서 두개 파일이 동일한 절대경로이름을 가질수는 없다. 만일 이름이 같은 두개 파일이 있다면 그것들은 서로 다른 등록부에 있어야 한다. 이것은 경로이름들이 서로 다르다는것을 의미한다. 즉 /home/romeo/progs/count.p1파일이 home/romeo/safe/count.p1파일과 함께 동시에 존재할수 있다.

참고

/home/romeo/pwgs와 같이 절대경로이름을 사용하여 파일을 언급하는 쉘스크립트를 쓰는 경우에는 홈등록부를 표현하는 구성요소를 HOME변수로서 즉 $HOME/progs와 같이 바꿀수도 있다. home등록부가 /u2/romeo인 다른 체계에서도 HOME변수가 항상 자기의 홈등록부를 알아내기때문에 $HOME/romeo지령은 여전히 동작한다.

지령에서 절대경로이름 사용하기

UNIX에서 지령은 디스크파일을 실행함으로써 실행된다. date지령을 주면 체계는 PATH변수에서 등록부목록으로부터 date파일을 찾아 낸 다음에 지령을 실행한다. 만일 지령의 위치를 알고 있다면 직접 경로이름과 함께 지령을 줄수 있다. date파일이 /bin등록부안에 있으면 절대경로이름을 사용할수 있다.

```
$ /bin/date
Tue Feb 15 12:18:37 EST 2000
```

누구도 date지령을 우와 같이 실행시키지 않는다. PATH변수로 지정된 등록부안에 있는 지령에 대해서는 절대경로이름사용이 필요 없다. PATH를 재호출해도 PATH는 /bin등록부와 /usr/bin을 자기목록안에 가진다. 대부분의 UNIX지령들은 이 등록부안에 갖추어 져 있다.

주해

만일 PATH에 없는 등록부에 있는 프로그람을 실행하려면 절대경로이름을 사용해야 한다. 례를 들어 /usr/rocal/bin등록부에 들어 있는 less프로그람을 실행하기 위해서는 절대경로이름 /usr/rocal/bin/less를 주어야 한다.

```
/usr/local/bin/less
```

만일 어떤 등록부안에 있는 프로그람에 자주 접근한다면 PATH에 등록부자체를 포함시키는것이 더 낫다. 이에 대해서는 17장에서 론의하기로 한다.

6.7 등록부의 변경(cd)

cd(change directory)지령을 사용함으로써 파일체계에서 등록부들사이로 이동할수 있다. 이 지령은 인수와 함께 사용될 때 인수로 지적된 등록부에로 현재등록부를 변경시킨다. progs등록부로 변경시키기 위하여 cd progs를 사용하자.

cd를 사용하기전과 사용한후에 두번 다 pwd로서 현재등록부를 검사한다.

```
$ pwd /home/romeo
$ cd progs              progs는 현재등록부로 되여야 한다
$ pwd
/home/romeo/progs
```

pwd는 절대경로이름을 표시하지만 cd는 절대경로이름사용을 요구하지 않는다. cd progs지령은 《현재등록부밑에 있는 progs등록부에로 보조등록부를 변경시키시오.》를 의미한다. 절대경로이름을 사용하여도 손해는 없다. 즉 cd/home/romeo/progs지령을 사용해도 그 효과는 같다. 만일 progs등록부가 자기 아래에 scripts등록부를 포함하고 있다면 home등록부로부터 scripts등록부로 변화시키기 위해서 다음의 지령을 사용해야 한다.

```
cd progs/scripts              progs등록부가 현재등록부안에 있다
```

또한 다음지령으로 scripts등록부안에 있는 convert.sh파일을 볼수 있다.

```
cat progs/scripts/convert.sh
```

여기서 경로이름들은 파일들사이에 /을 가지지만 그것들은 절대경로이름이 아니라 상대경로이름이다. 그러나 /bin등록부로 변화시키기 위해서는 절대경로이름이 필요하다.

```
$ pwd
/home/romeo/progs
$ cd /bin               bin이 현재등록부에 없기때문에
$ pwd           절대경로이름이 요구된다
/bin
```

또한 상대경로이름을 사용하여 /bin(혹은 어떤 등록부)에로 이동할수 있다. 절대경로이름을 사용하는 cd지령에 의한 이동을 그림 6-2에서 보여 주고 있다.

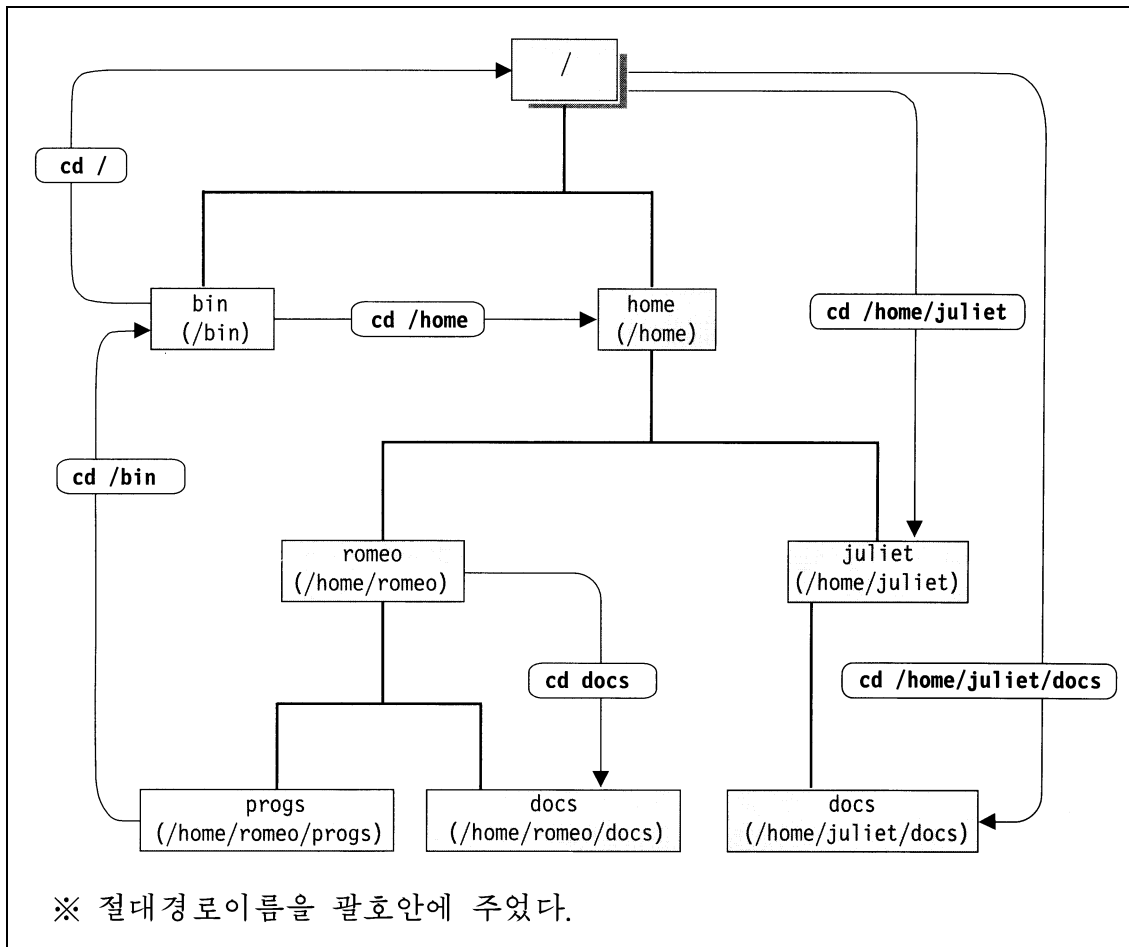



그림 6-2. cd지령에 의한 항행

인수없이 cd를 사용하기

cd지령은 인수없이 사용될 때 특수한 기능을 수행한다.

```
$ pwd
/home/romeo/progs
$ cd
$ pwd
/home/romeo                인수없이 사용된 cd는 홈등록부로 되돌아 간다
```

Windows사용자들을 주의해야 한다! 인수없이 호출된 이 지령은 현재등록부를 지적하지 않는다. 사용자가 원래 가입한 등록부에로 홈등록부를 절환하므로 간단히 cd를 사용하여 홈등록부에 직접 되돌아 갈수 있다.

```
$ cd /home/juliet
$ pwd
/home/juliet
$ cd             홈등록부로 돌아 간다
$ pwd
/home/romeo
```

만일 등록부접근허가권을 가지지 못했다면 cd지령은 때때로 실패할수 있다. 만일 등록부허가권을 침착하게 변경하면 이런 현상은 생기지 않는다. 파일과 등록부허가는 다음장에서 론의된다.

주해

cd가 인수없이 호출되면 자기의 홈등록부로 쉽게 되돌아 간다. cd는 현재등록부를 보여 주지 않는다.

6.8 상대경로이름(.과 ..)

앞의 실례에서 절대경로이름과 cd를 사용하여 /home/romeo등록부로부터 /home/juliet등록부에로 등록부를 변경하였다.

```
cd /home/juliet
```

현재등록부와 같은 어미등록부(/home)를 가지는 등록부에로 이동하는것은 어렵기때문에 절대경로이름을 사용해야 한다. 그러면 현재등록부에서 /home등록부로 어떻게 이동해야 하는가? cd/home을 사용하는 방법은 절대경로이름이 길수록 지령이 길어 지기때문에 적당치 못하다.

이것을 위해서 UNIX는 지름길 즉 상대적인 경로이름을 제공한다. 앞절에서 그 형태를 기본적으로 보았지만 아래의 2개 기호들을 사용하는것이 가장 효과적이다.

- .(한개의 점)-이것은 현재등록부를 표시한다.
- ..(두개의 점)-이것은 어미등록부를 표시한다.

상대경로이름을 조작하기 위해서 ..을 사용하자. 어미등록부로 이동하기 위해서는 cd..를 사용해야 한다.

```
$ pwd
/home/romeo/progs
$ cd ..
$ pwd
/home/romeo              한 준위우로 이동
```

이 방법은 정밀하며 계층구조로 올라 갈 때 더 쓸모 있다. cd ..지령은 사용자의 등록부를 현재등록부의 어미등록부에로 변경하라는것으로 해석한다.

..역시 경로이름의 구성요소를 이룬다. /을 리용하여 ..을 여러번 쓸수 있다. 그러나 /이 ..과 함께 사용될 때 서로 다른 의미를 초래한다. 즉 한 준위아래로 이동하는것대신에 한 준위우로 이동한다. 실례로 /home등록부로 이동하기 위해서 cd/home을 사용하였는데 cd/hom대신에 상대경로이름을 사용할수 있다.

```
$ pwd
/home/romeo/progs
$ cd ../..               두개 준위우로 이동
$ pwd
/home
```

이제 /home/romeo로부터 /home/juliet로 이동하기 위해서 어떻게 해야 하는가? 이것은 경로이름안

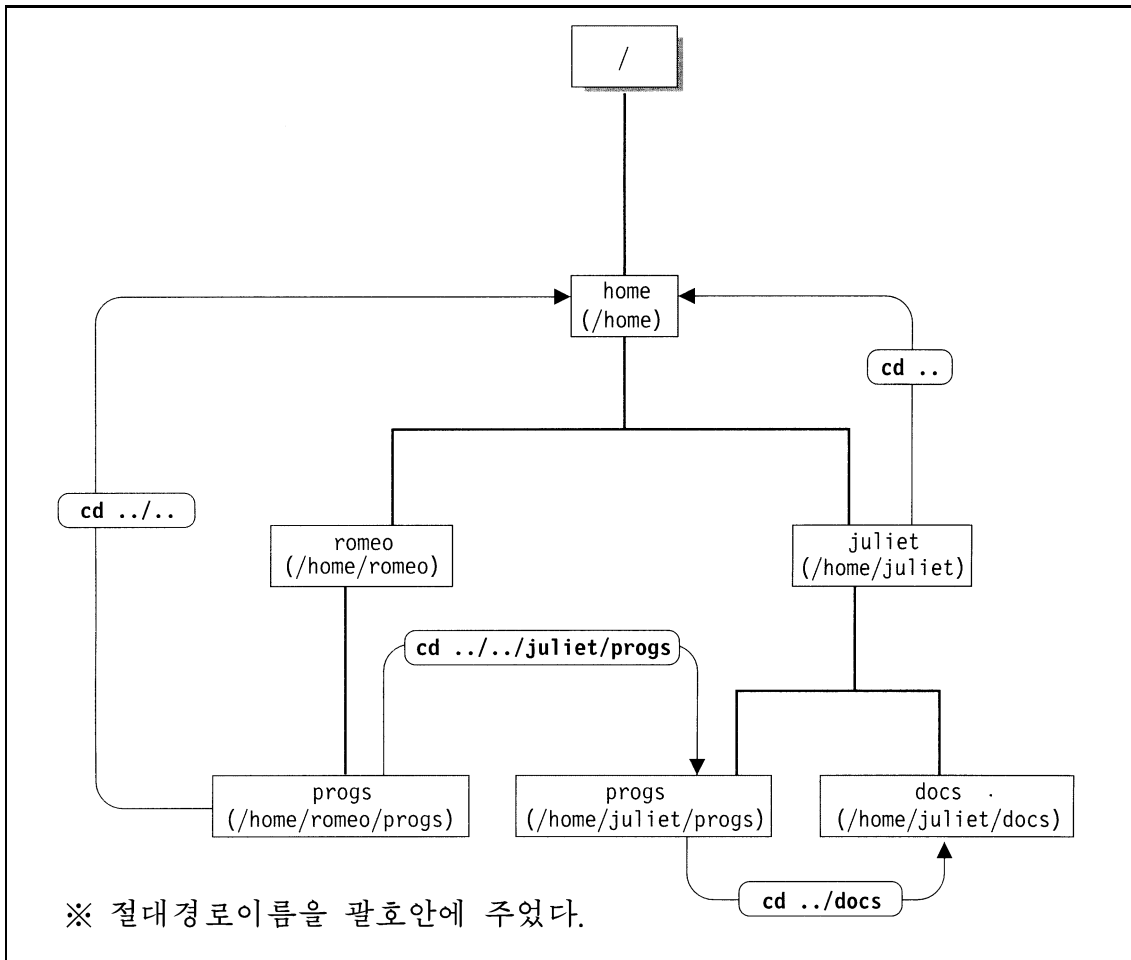



그림 6-3. 상대경로이름에 의한 항행

에 ..과 목적등록부이름의 결합을 요구한다.

```
$ pwd
/home/romeo
$ cd ../juliet          한 준위우로 그다음 아래로
$ pwd
/home/juliet
```

..을 사용하고 있는 상대경로이름의 사용을 그림 6-3에서 보여 준다.

인수로서 현재등록부를 사용하는 어떤 지령은 한점을 가지고 조종할수 있다. 이것은 마지막인수로서 등록부를 사용하고 있는 cp지령이 한개 점으로 사용될수 있다는것을 의미한다.

```
cp ../juliet/pricelist.html .
```

이것은 현재등록부(.)에 pricelist.html파일을 복사한다. 복사되는 파일이름을 주지 않았으므로 복사된 파일이름 역시 원본파일이름과 같다.

주의

../..을 사용할 때 /의 오른쪽에 있는 두개 점은 /의 왼쪽에 있는 두개 점으로 렬거된 등록부의 어미등록부를 나타낸다.

때때로 지령앞에 ./을 놓는것이 필요할것이다. 현재등록부가 사용자가 작성한 cat프로그람을 가지고 있다고 가정하자. 여기서 cat프로그람은 /bin등록부에 있는 UNIX cat와 다르다. 이제 cat note지령은 PATH에서 .보다 /bin등록부가 더 빨리 발생하므로 /bin안에 있는 cat지령을 실행할것이다(echo $PATH지령을 사용하여 그것을 확인해 보시오). /을 사용하면 사용자가 작성한 cat지령을 실행시키고 /bin등록부에 있는 cat지령은 무시할수 있다.

```
./cat note                현재등록부에 있는 cat
```

여기서 현재등록부안에 지령과 인수가 둘 다 있는것을 주의해서 보시오.

주해

절대경로이름을 쓰겠는가 상대경로이름을 쓰겠는가 하는것은 경로이름을 서술하는데 요구되는 건누르기회수와 관련된다. 대체로는 상대경로이름이 보다 적은 건누르기를 실현하지만 사용자가 현재 어디에 놓여 있는가에 따라 절대경로이름이 보다 적은 건누르기를 실현하는 경우도 있다.

6.9 등록부의 만들기(mkdir)

등록부들은 mkdir지령으로 만든다. 지령뒤에는 만들어야 할 등록부이름이 따른다. patch등록부는 지령 mkdir patch에 의하여 현재등록부밑에 만들어 진다. mkdir는 다중인수들을 가진다. 즉 한개의 mkdir지령으로서 몇개의 보조등록부들을 만들수 있다.

```
mkdir patch dbs doc            3개의 등록부를 만든다
```

UNIX체계는 더 나아가서 지령을 한번 호출하여 등록부나무를 만들수 있게 한다. 실례로 다음의 지령은 등록부구조를 창조한다.

```
mkdir pis pis/progs pis/data            등록부나무를 만든다
```

이것은 3개의 보조등록부 즉 pis등록부와 pis등록부안에 2개의 보조등록부를 만든다. 인수를 렬거하는 순서가 중요하다. 즉 어미등록부를 만들기전에 보조등록부를 절대로 만들수 없다. 실례로 다음과 같이 지령을 줄수 없다.

```
$ mkdir pis/data pis/progs pis
mkdir: can't access pis/.
mkdir: can't access pis/.
```

우의 통보문은 pis등록부를 만들지 않았기때문에 progs와 data인 두개의 보조등록부에 대한 만들기가 실패했다는것을 통지한다. 때때로 체계는 다음의 경우에 등록부만들기를 거부한다.

```
$ mkdir test
mkdir : Failed to make directory "test"; Permission denied
```

그 원인은 다음과 같다.

- test등록부가 이미 존재할 때
- 현재등록부에서 이름이 보통파일로 되였을 때
- 현재등록부에 대한 허가권모임이 사용자에 의한 파일과 등록부의 만들기를 허락하지 못할 때

다음장에서 파일허가권에 대하여 취급하게 된다.

6.10 등록부의 삭제 (rmdir)

rmdir지령은 등록부를 삭제한다. pis등록부를 삭제하기 위해서는 지령 `rmdir pis`를 사용해야 한다. mkdir와 같이 rmdir도 한번에 여러개의 등록부를 삭제할수 있다. 실례로 mkdir지령으로 만든 3개의 등록부들과 보조등록부들은 거꾸로 설정된 인수들로서 rmdir지령을 사용하여 제거될수 있다.

```
rmdir pis/data pis/progs pis            이때 인수들은 반대로 쓴다
```

등록부와 그의 보조등록부들을 지울 때 반대론리가 적용된다. mkdir지령에서 사용되는 등록부렬은 rmdir에서는 무효로 된다. 즉

```
$ rmdir pis pis/progs pis/data
rmdir: pis: Directory not empty
```

오유통보문으로부터 무엇을 알수 있는가? rmdir지령은 더 낮은 준위의 보조등록부들인 progs와 data를 삭제하였다. 등록부들을 삭제할 때 두가지 규칙을 반드시 명심해야 한다.

- rmdir지령은 텅 빈 등록부만 삭제할수 있다. 우의 경우에는 pis등록부가 자기밑에 progs와 data 보조등록부를 가지고 있으므로 제거될수 없다.
- 보조등록부를 제거하기 위해서는 반드시 그 등록부의 웃준위등록부에서 제거해야 한다(이 제한은 Linux에서 적용되지 않는다.).

첫번째 규칙은 우의 실례에 귀착된다. 하지만 보통파일들을 가지고 작업하는 UNIX rm지령은 완전한 등록부구조를 제거하기 위해 특수선택항목(-r)을 사용할수 있다. 이것은 6.12에서 론의된다. 두번째 규칙을 리해하기 위하여 progs등록부자체에서 지령을 실행하여 progs등록부를 제거해 보자.

```
$ cd progs
$ pwd
/home/romeo/pis/progs
$ rmdir .                UNIX에서 작업하는 현재등록부를 제거하기
rmdir: .: Can't remove current directory or ..
```

progs등록부를 제거하기 위해서는 progs우에 있는 등록부 다시 말하여 pis등록부에서 제거해야 한다.

```
$ cd ..          어미등록부로 이동
$ pwd
/home/romeo/pis
$ rmdir progs
$ _              성공적인 지령
```

mkdir와 rmdir지령은 오직 사용자가 소유한 등록부에서만 실행된다. 일반적으로 사용자는 자기 홈등록부의 소유자이며 그는 홈등록부 혹은 자기가 만든 임의의 보조등록부안에서 보조등록부(파일들뿐아니라)들을 만들거나 제거할수 있다. 그러나 표준적으로는 다른 사용자들의 등록부안에서 파일과 등록부들을 만들거나 제거할수 없다. 소유권에 대한 개념은 다음장에서 론의된다.

주해

제거하려는 보조등록부가 텅 비지 않았거나 사용자가 그의 어미등록부에 위치하지 않았을 때는 rmdir지령을 사용할수 없다. 그러나 rm지령을 리용하면 등록부가 비여 있든 비여 있지 않든 관계없이 제거할수 있다. Linux인 경우에는 rmdir .지령을 리용하여 현재등록부를 삭제할수 있다(그것이 비여 있는 경우에만).

6.11 파일복사(cp)

cp지령은 파일 또는 파일들의 묶음을 복사한다. 이 지령은 서로 다른 이름으로 디스크상에 꼭 같은 파일을 만든다. 문장구성에서 적어도 2개의 파일이름이 지정되여야 하며 그 파일들이 보통파일들인 경우에는 첫번째 파일이 두번째 파일에 복사된다.

```
cp chapl unitl
```

만일 목적파일(unitl)이 존재하지 않는다면 그 목적파일이 만들어 지며 이미 존재하고 있다면 체계가 예고없이 덧쓰기한다. 그렇기때문에 목적파일이름을 선택할 때 주의해야 한다. 파일이 존재하고 있는가 없는가 하는 것을 ls지령으로 검사해 볼수 있다.

만일 unitl이 존재하고 보통파일이 아니라 등록부파일인 경우에는 chapl이 같은 이름으로 unitl에 복사된다.

```
cp chapl unitl/chap1          cp chap1 until1과 같다
```

cp지령의 목적파일에 등록부이름을 지정하여 그 등록부안에 여러개의 파일들을 복사할수 있다.

/home/juliet 등록부에 있는 파일 .profile을 현재등록부에 복사하기 위하여 cp지령에 .(점)을 리용할수 있다. 실례로 /home/juliet로부터 현재등록부에 .profile파일을 복사하기 위하여 다음의 지령들가운데서 한 지령을 사용할수 있다.

```
cp /home/juliet/.profile .profile          목적은 파일이다
cp /home/juliet/.profile .          목적은 현재등록부이다
```

첫번째 지령은 건누르기를 더 요구하므로 두번째 지령을 사용하는것이 더 낫다. 앞에서 론의된바와 같이 cp지령을 리용하여 한개이상의 파일을 복사할수 있다.

```
cp chap01 chap02 chap03 progs          progs는 등록부가 되여야 한다
```

이 파일들은 progs등록부안에 같은 이름으로 복사된다. 만일 이 파일들이 progs등록부안에 이미 있었다면 덧쓰기된다. 우의 지령이 실행되려면 progs등록부가 존재하고 있어야 한다.

UNIX체계는 여러개의 파일을 정합하기 위하여 **메타문자**(metacharacter)라고 부르는 특수한 문자들의 모임을 리용한다. 자세한 론의는 8장에서 계속하겠지만 여기서는 메타문자모임의 한 문자인 * 에 대하여 고찰하겠다. 만일 현재등록부안에 chap로 시작되는 3개 파일이 있다면 chap뒤에 *를 쓸수 있다.

```
cp chap* progs          chap로 시작되는 모든 파일들을 복사한다
```

다중파일이름들이 같은 문자렬을 공유하는 경우에는 *를 간략기호로 사용한다. 메타문자의 사용에 대해서는 8.2에서 론의된다.

주의

cp지령은 목적파일이 존재하면 예고없이 덧쓰기한다. cp지령을 리용하기전에 ls지령으로 목적파일이 존재하는가를 검사하고 다음에 cat지령으로 파일의 내용을 검사하시오.

cp의 선택항목

대화형복사(-i)

-i(interactive)선택항목은 목적파일을 덧쓰기하기전에 사용자에게 예고한다. 만일 unitl파일이 존재

하고 있다면 cp지령은 응답을 위한 프롬프트를 준다.

```
$ cp -i chap1 unitl
cp: overwrite unit1? y          unitl파일은 여기서 보통파일이다
```

y로 응답하면 파일을 덧쓰기한다. y로 응답하지 않으면 파일을 복사하지 않는다.

만일 사용자가 잘못하여 파일을 덧쓰기했다면 별명(17.4)과 쉘함수(19.10)를 리용할수 있다.

등록부구조복사(-r)

-r(recursive)선택항목을 리용하여 등록부구조를 통채로 복사할수 있다. 아래의 지령은 newprogs 등록부에 progs등록부안에 있는 파일들과 보조등록부들을 모두 복사한다.

```
cp -r progs newprogs
```

cp지령이 보조등록부안에 있는 모든 파일들을 복사하기때문에 보조등록부가 없다면 만들어야 한다. 또한 이 방법으로 등록부구조를 통채로 이동시킬수 있다.

주해

만일 복사되여야 할 파일이 읽기보호되였다면 복사할수 없다. cp지령은 또한 목적파일이 존재하고 있고 쓰기보호되였을 때 실행되지 않는다. 이 문제는 다음장에서 취급된다.

6.12 파일삭제(rm)

rm지령은 파일들을 지우며 디스크상에 쓸수 있는 공간을 만든다. 이 지령은 주의해서 리용해야 한다. 한개의 명령으로 여러개의 파일을 삭제할수 있다.

```
rm chap01 chap02 chap03          rn chap*로도 할수 있다
```

만일 -r선택항목을 리용하지 않으면 등록부는 제거할수 없지만 그 등록부에 있는 파일들은 제거할수 있다. rm지령에 cd를 리용하지 않고도 progs등록부로부터 2개의 파일들을 제거할수 있다.

```
rm progs/chap01 progs/chap02            혹은 rm progs/chap[12]
```

지우기조작의 부분으로서 때때로 등록부안에 있는 모든 파일들을 지울 필요가 있을 때 rm지령에 모든 파일들을 표시하는 *를 리용할수 있다.

```
$ rm *          모든 파일들을 지웠다
$ _
```

이 방식으로 파일들을 지울 때 체계가 파일들을 제거하기에 앞서 Are you sure? 또는 All files in directory will be deleted라는 통보문을 주지 않기때문에 Windows사용자들은 주의하여 이 지령을 사용해야 한다. $프롬프트가 나타나면 지령이 실행되였다는것을 의미한다.

주의

pwd지령으로 현재등록부를 검사하고 ls로 파일들을 렬거해 본 다음에 rm *로 지령을 써야 한다.

rm의 선택항목

대화형삭제(-i)

때때로 일부 파일은 그대로 놔두고 파일들을 지워야 할 경우가 있을수 있다. 만일 20개 파일을 지워야 한다면 지령행에 파일들의 이름을 모두 지정하지 않아도 된다. 이런 경우에는 매개 파일들을 삭제하기에 앞서 사용자에게 확인을 요구하는 -i선택항목을 리용할수 있다. 즉

```
$ rm -i chap01 chap02 chap03
chap01: ?y
chap02: ?n
chap03: ?y
```

y는 파일을 제거하며 n은 파일을 지우지 않는다.

재귀적(그리고 위험한)삭제(-r)

-r선택항목을 가지고 rm은 보조등록부안에 있는 파일들과 모든 보조등록부들에 대한 완전한 재귀적탐색을 진행한다. 다음에 그 모든것들을 지운다.

rm지령은 표준적으로 등록부들을 제거하지 못하지만 -r선택항목을 리용하면 등록부를 제거한다. 따라서 지령

```
rm -r *          현재등록부나무가 완전히 제거되였다
```

을 주면 모든 보조등록부와 그안의 파일들은 물론 현재등록부안에 있는 모든 파일들을 삭제한다. rm지령을 실행하기에 앞서 지령을 두세번 2중으로 확인해 보아야 한다. 그렇지 않으면 rm *지령을 주는것보다 후과가 더 참혹해 진다.

혹시 중요한 파일이 남아 있지 않는가를 확인하기 위하여 그것의 모든 파일들과 등록부들의 목록을 보고 현재등록부를 검사해야 한다. 이 파일을 여벌로 복사하지 않는다면 영원히 잃어 버리게 될것이다.

rm은 쓰기보호된 파일들을 삭제할수 없다. -f선택항목을 리용하여 쓰기보호된 파일도 삭제할수 있다.

```
rm -rf *
```

가지고 있지 않는 파일은 일반적으로 삭제할수 없다. rm지령으로 제거된 파일들은 휴지통에 보내지지 않고 완전히 지워 진다.

주의

rmdir지령과 달리 rm -r지령은 비지 않은 보조등록부들을 제거할수 있다. 만일 뿌리사용자(상급사용자)가 뿌리등록부에서 rm -rf*지령을 실행하면 하드디스크에서 전체 UNIX체계가 지워진다.

6.13 파일이름고치기(mv)

mv지령은 파일과 등록부이름을 다시 짓는다. 이 지령은 파일을 복사하는것이 아니라 단지 이름만 다시 짓는다. 이것은 이 지령이 디스크공간을 소비하지 않는다는것을 의미한다. 이 지령은 두가지 기능을 수행한다.

- 파일(또는 등록부)이름고치기

- 파일묶음을 다른 등록부에 이동하기

chap01파일이 보통파일일 때 이름을 고쳐 짓기 위해서는 지령

```
mv chap01 man01          man01은 등록부일수도 있다
```

을 사용해야 한다. man01은 또한 등록부가 될수 있다. 만일 목적파일이 존재하지 않는다면 이름이 고쳐질것이다. 목적파일이 존재하고 있다면 기정적으로 덧쓰기를 진행하므로 주의해서 지령을 리용해야 한다.

cp지령과 마찬가지로 파일묶음은 오직 한개 등록부에만 이동된다. 아래의 지령은 progs등록부에 3개 파일을 이동한다.

```
mv chap01 chap02 chap03 progs          progs는 등록부여야 한다
```

mv지령은 또한 이름을 다시 쓰기할수 있는데 리용방법은 보통파일인 때의 경우와 같다. mv에도 cp에서와 마찬가지로 -j선택항목이 쓰인다. 이 항목을 써서 파일이 덧쓰기되지 않도록 할수 있다.

비록 cp, rm과 mu지령들은 문장구성이 간단하지만 절대 및 상대경로이름들과 함께 리용된다. 표 6-1에서 이 지령들에서 조종되는 인수들의 종류를 보여 주었다.

표 6-1.　　　　cp, rm, mv지령들의 사용법

지령행	동 작
cp note ..	어미등록부에 note파일을 복사한다
cp ../note .	어미등록부에서 현재등록부에 note파일을 복사한다
rm ../bar/index	현재등록부와 같은 계층위치에서 bar등록부에 있는 index파일을 지운다
mv foo1 foo2 /foo1/foo2	/foo1/foo2등록부에 foo1과 foo2파일을 이동시킨다
rm -r bar	bar의 등록부구조를 전부 지운다. 만일 bar가 보통파일이면 bar파일만 지운다
cp -r . ../bar	어미등록부밑에 있는 bar등록부에 현재등록부구조를 복사한다(bar등록부가 존재할 때만)
mv ../* .	어미등록부에서 현재등록부에 모든 파일을 이동한다

6.14 파일의 현시와 만들기(cat)

파일들의 복사와 제거에 앞서 파일내용을 확인해야 할 경우가 있다. cat지령은 말단에 파일의 내용을 표시하는 UNIX체계지령이다.

```
$ cat note1
This is a file containing simply this sentence.
```

1.10.4에서 echo로 만든 파일을 보기 위해 cat지령을 사용하였다. 많은 UNIX지령들과 마찬가지로 cat지령도 한개이상의 파일이름을 인수로 사용한다.

```
cat chap01 chap02
```

이 지령은 첫번째 파일다음에 두번째 파일의 내용을 어떤 머리부정보가 없이 직접적으로 보여 준다. 여기서 cat지령은 두개 파일을 련결(concatenate)하고 있는데 그의 이름도 여기서 유래되였다. 한개 파

일을 현시하는것은 련결의 특수경우일뿐이다.

cat지령은 표준적으로 본문파일만 표시한다. 만일 입력에 인쇄할수 없는 ASCII문자들이 있다면 이 문자들을 표시하기 위해서 cat지령에 -v선택항목을 사용할수 있다.

cat지령을 리용한 파일만들기

cat지령은 또한 파일을 만들기 위해 리용된다. vi와 같은 편집기를 리용하지 않고 작은 파일을 만드는 방법을 알아야 한다.

```
$ cat>foo                이때 파일이 만들어 진다
A > symbol following the command means that the
output goes to the filename following it. cat used
in this way represents a rudimentary editor.
[Ctrl-d]                 파일의 끝
$ _              프롬프트는 돌려 준다
```

지령행을 입력하고 [Enter]건을 누르면 프롬프트가 없어 지고 사용자로부터 입력을 대기한다. 3개 행을 입력할 때 매행을 [Enter]건으로 분리시키면 된다. 마지막으로 체계에 입력이 끝났다는것을 지정하기 위해서 [ctrl-d]를 눌러야 한다. UNIX에서는 파일의 끝을 표시하기 위해 [ctrl-d]문자를 사용한다. 1장과 3장에서 이 문자의 사용과 이 문자를 변경시키는 stty지령에 대하여 언급하였다. 이 문자를 입력하면 체계는 본문입력이 끝난것으로 리해한다.

따라서 foo파일이 만들어 지며 프롬프트를 돌려 준다. 《cat》지령으로 이 파일을 확인해 보자.

```
$ cat foo                파일을 보려고 하는 경우에는 >가 필요없다
A > symbol following the command means that the
output goes to the filename following it. cat used
in this way represents a rudimentary editor.
```

주해

[Ctrl-d]문자는 cat지령에서만이 아니라 건반으로부터 입력을 요구하는 모든 지령에서 리용된다. bc지령(3.12)에서도 이 문자가 사용된다.

cat지령은 또한 파일들에 대한 추가도 진행한다. 인수로서 파일이름을 렬거하는것으로써만이 아니라 또 다른 지령의 출력으로부터 cat에 입력을 제공할수 있다. 출력은 반드시 말단에 가지 않아도 되며 대신 파일에 갈수도 있다. 8장에서 이 모든것에 대해서 배우게 될것이다.

6.15 파일형알아보기(file)

3가지 형의 파일 즉 보통파일, 등록부파일 그리고 장치파일에 대해서 더 구체적으로 알 필요가 있다. 실례로 정규파일은 간단한 본문, C프로그람 또는 실행가능한 코드를 포함한다. 정규파일이 포함하고 있는 자료의 형을 알아 내기 위해서 cat지령을 사용하는 대신 file지령을 사용할수 있다. file지령은 파일의 《보조형》을 기록한다.

```
$ file emp.lst
emp.lst: ascii text          본문파일
```

file지령은 넓은 범위에서 정보목록을 표시한다. 출력은 체계에 의존하는데 일부 체계는 다른 체계들보다 파일을 더 잘 식별할수 있다. 다음의 지령에서 모든 파일들을 표시하기 위해 *가 사용된다.

```
$ file *
addressbook.gif        GIF image data, version 87a, 482 x 345
backup.sh              executable /bin./ksh script
basic1.html            HTML document text
cmeri_dns.tar:         GNU tar archive
dialout.sh             executable shell script
hosts.Z:               compress'd data 16 bits
ux3rd.gz:              gzip compressed data
wins.zip               Zip archive data
wstovi.c:              C program text
```

이 지령은 파일의 형을 문맥에 의해 식별하는 내부기구를 가지고 있다. 또한 tar보존파일(archive), 화상들, HTML파일들과 압축된 파일들을 식별한다.

6.16 파일인쇄(lp, cancel)

사용자는 인쇄기에 직접 접근하지 못하게 되여 있다. 그 대신에 일감을 인쇄대기렬에 다른것들과 함께 줄 세워 놓아야(spool:완충)한다. **완충**(spooling)은 일감들이 순서대로 인쇄되게 하며 사용자가 인쇄자원의 리용을 관리할 필요가 없게 한다. System V에서 완충기능은 lp(line printing)지령에 의해 제공된다.

다음의 lp지령은 chap 01파일을 인쇄한다.

```
$ lp chap01
request id is prl-320 (l file)
$ _
```

일감이 제출된후 즉시에 프롬프트가 귀환된다. 파일은 실지로 지령이 호출된 때에 인쇄되지 않고 그 후에 인쇄대기렬에 완충된 일감수에 따라 인쇄된다. 여러명의 사용자들이 이런 방법으로 충돌이 없이 파일들을 인쇄할수 있다.

lp는 요청식별자(request-id) 즉 인쇄기이름(prl)과 일감번호(320 즉 다른 지령으로 후에 호출될수 있는)를 통지한다. 흔히 파일의 경복사(hard copy)에 앞서 사용자이름, 요청식별자, 날자에 관한 제목페지가 선행한다.

6.16.1 lp의 선택항목

lp지령은 여러 선택항목들을 리용하는데 체계에 따라서 선택항목들이 서로 다르다. 일부 공통된 선택항목들을 아래에서 소개한다.

앞의 실례에서 lp지령은 기정인쇄기가 관리자에 의해 정의되였기때문에 요청을 접수하였다. 만일 기정인쇄기가 아니거나 혹은 체계에 인쇄기가 1대이상 있다면 인쇄기이름(실례로 laser)과 함께 -d선택항목을 사용해야 한다.

```
lp -dlaser chap01            또한 -d다음에 공백을 줄수도 있다
```

제목문자렬의 앞에 놓이는 -t(title)선택항목은 첫 페지에 제목을 인쇄한다

```
lp -t"First chapter" chap01           -t뒤에 공백을 줄수 있다
```

-m(mail)선택항목을 사용하여 파일이 인쇄된 다음에 사용자에게 통지할수 있으며 -n선택항목을 리용하여 파일을 여러번 인쇄할수 있다.

```
lp -n3 -m chap01             파일을 3번 인쇄하고 통지한다
```

lp지령에 파일이름이 항상 리용되지는 않는다. UNIX체계는 관흐름(제8장)의 개념을 리용하여 이 지령들에 입력으로 작용하는 또 다른 지령의 출력을 허가한다.

6.16.2 일감의 취소(cancel)

lp지령에 의해 대기상태에 들어 선 일감들은 인수로 일감의 요청식별자 혹은 인쇄기이름을 사용하고 있는 cancel지령에 의해 취소된다.

```
cancel laser           인쇄기 laser상에서 현재일감을 취소한다
cancel prl-320         요청식별자가 prl-320인 일감이 취소된다
```

주해

사용자는 자기가 가지고 있는 일감들만 취소하지만 체계관리자는 어떠한 일감이라도 취소할수 있다.

Linux

lpr, lpq, lprm지령으로 인쇄하기

Linux는 인쇄를 위한 lpr지령과 일감들을 제거하는 lprm지령을 제공하고 있는 버클리의 인쇄체계를 사용하고 있다. 이 지령은 일반적으로 일감번호를 내지 않는다.

```
$ lpr typescript
$ _
```

일감번호를 알려면 lpq지령을 사용하시오.

```
$ lpq
lp is ready and printing
Rank     Owner   Job     Files         Tatal Size
active   sumit   17      typescript    3615 bytes
```

만일 이 일감을 대기렬에서 제거하려면 다음명령을 사용해야 한다.

```
$ lprm 17
dfA017saturn dequeued              17번째 일감이 제거된다
cfA017saturn dequeued
```

또한 인수로 이음표(-)를 사용함으로써 사용자가 소유한 모든 일감들을 제거할수 있다.

```
lprm -             사용자가 소유한 모든 일감들을 제거
```

사용자는 System V의 lp와 마찬가지로 복사본들을 지정된 수만큼 인쇄할수 있으며 제목을 고를수 있으며 특정한 인쇄기에 출력을 직접 가리킬수 있다. 또한 일감의 완료를 통지할수 있다.

```
lpr -p dj500 foo           인쇄기 dj500에 인쇄한다
lpr -T "The List of RFCs" foo         이 제목을 사용
lpr -#3            복사물 3개 인쇄
lpr -m foo            완성후에 통보문을 내보낸다
```

6.17 디스크의 빈 공간찾기(df)

UNIX파일체계가 한개의 뿌리등록부를 가진 나무구조처럼 고찰되지만 실지에 있어서는 전혀 다르다. 여러개의 디스크들가운데서 매개 디스크는 자기의 뿌리등록부와 함께 적어도 한개의 **파일체계**(file system)를 가지고 있으며 때때로는 여러개의 파일체계를 가지고 있다. 시동렬(booting sequence)은 이 모든 나무구조들을 한개로 통합하며 마치도 한개의 파일체계가 있는것처럼 보여 준다. 그러나 이것들은 명백히 다른 구조들이며 빈 공간을 다른것이 리용할수 없다.

다중파일체계의 개념과 그것들의 조종에서는 21장에서 취급된다. df지령을 사용할 때를 제외하고는 그것들을 한개의 실체처럼 리해하면 될것이다. 디스크상에 쓸수 있는 빈 공간을 표시하기 위해서는 df지령을 사용해야 한다. 출력은 항상 매개 파일체계를 따로따로 통지한다.

```
$ df
/          (/dev/dsk/c0t0d0s0 ): 4122488 blocks    446163 files
/proc      (/proc             ):       0 blocks     15861 files
/dev/fd    (fd                ):       0 blocks         0 files
/d01       (/dev/dsk/c0t8d0s0 ): 3197798 blocks    667713 files
/d02       (/dev/dsk/c0t8d0s1 ): 8587618 blocks    614579 files
/d03       (/dev/dsk/c0t8d0s3 ): 6435158 blocks    614547 files
/oracle    (/dev/dsk/c0t0d0s3 ): 6018636 blocks    495356 files
/tmp       (swap              ): 3649776 blocks    170946 files
```

8개의 파일체계가 있는데 일부는 체계를 설치할 때 만들어 진다. 뿌리파일체계(첫번째 렬에 있는 /)는 4,122,488개의 블로크라는 빈 디스크공간을 가지고 있다. UNIX지령의 대부분은 블로크단위로 디스크공간을 통지하는데 1개 블로크크기는 보통 512byte이다.

뿌리파일체계는 또한 446,163개의 **색인마디**(inode)들을 가지는데 이것은 이 체계상에 많은 파일들이 추가될수 있다는것을 의미한다(일부 체계들은 마지막렬에서 "files"보다도 "inodes"를 보여 주고 있다). System V체계는 블로크 혹은 색인마디가 점차적으로 없어 질 때까지 기능을 계속 수행할것이다. 체계에서 전체 빈 공간은 8개의 파일체계에 대한 빈 공간을 모두 합한것이다.

-t선택항목은 매 파일체계에서 전체 디스크공간을 알아 내기 위해 사용되는데 다음의 지령은 orcale 파일체계의 사용공간만을 표시한다.

```
$ df -t /oracle
/oracle    (/dev/dsk/c0t0d0s3 ):    6018636 blocks     495356 files
                         total:     6018654 blocks     495360 files
```

간단히 해설하면 이 파일체계의 전체 사용공간은 파일에 대해서는 6,018,654개의 블로크이며 색인마디에 대해서는 495,360이다. df지령은 감시를 목적으로 하고 있는 체계관리자에 의해서 정기적으로 사용된다.

주해

한개 파일체계에 있는 사용공간을 완전히 소비했을 때 그 파일체계는 다른 파일체계의 사용공간을 빌려 쓸수 없다

Linux

df지령은 각이한 출력을 현시하는데 여기서는 디스크공간을 퍼센트(%)로 보여 주며 블로크크기는 1024byte이다.

```
$ df
File system     1024-block  Used Available Capacity Mounted on
/dev/hdb3        485925   342168    118658    74%  /
```

Linux는 각이한 목적에 따라 -t선택항목을 사용하는데 기정출력만으로도 정보를 충분히 제공 받을수 있다.

6.18 디스크의 소비정형알아보기(du)

전체 파일체계의 소비구역이 아니라 지정된 등록부나무의 소비구역을 알아야 하는 경우가 제기될것이다. du(disk usage)지령은 등록부나무를 재귀적으로 검사하여 사용공간을 통지하는 지령이다. 다음의 지령은 /home/sales/tml등록부의 사용공간을 보여 준다.

```
$ du /home/sales/tml
11554    /home/sales/tm1/forms
12820    /home/sales/tml/data
638      /home/sales/tml/database
…
25170    /home/sales/tm1                 또한 마지막에 합계를 통지한다
```

기정적으로 du지령은 매 보조등록부의 사용공간을 보여 주며 마지막에 합계를 통지한다. 목록이 때때로 매우 커질수 있기때문에 -s(summary)선택항목을 리용하여 전체 보조등록부의 사용공간을 하나로 함축하여 보여 줄수 있다.

```
# du -s /home/sales/tml
25170   /home/sales/tml
```

항상 디스크공간에 대한 쟁탈전이 존재한다. 사용자들은 흔히 더이상 쓰지 않는 파일들을 지우지 않는다. 때문에 관리자는 매 사용자들의 홈등록부(22.10.1)에 사용된 공간을 감시하여야 한다. 체계관리자는 쉘스크립트에서 이 지령을 정기적으로 사용하여 유명한 디스크공간랑비자(disk hoggers)들을 찾아낼수 있다.

참고

-s선택항목을 사용하여 홈등록부의 사용공간을 알아 낼수 있다.

```
du -s /home/romeo
```

메타문자를 사용하여 사용공간을 사용자별로 알아 낼수 있다. 관리기능은 22.10.1에서 취급된다.

6.19 파일의 압축(compress, gzip, zip)

디스크공간을 보존하기 위해서 용량이 크거나 드물게 리용되는 파일들을 압축할수 있는데 이것들은 디스크용량을 적게 소비한다. 가장 많이 쓰이고 있는 압축프로그람은 compress, gzip, zip이다. 마지막 2개는 Linux체계상에서 일반적으로 쓰인다. compress와 gzip는 다중파일들을 다루며 압축후에 원래파일

을 제거한다. zip는 원래파일을 남겨 두며 여러 파일들을 하나의 파일(보존파일(archive)이라고 한다.)에 묶어 놓는다.

이 편의프로그람들에 의하여 압축할수 있는 압축의 정도는 사용한 압축편의프로그람과 압축하려는 파일의 형에 관계된다. Postscript, PDF와 본문파일들은 압축되지만 확장자가 .gif와 jpeg인 화상파일들은 그것들자체가 압축되는 형태에 도형자료를 가지기때문에 전혀 압축되지 않는다. 표 6-2에 압축의 정도를 형별로 보여 주었다.

표 6-2. 파일의 압축

파일형	압축(%)		
	compress	**gzip**	**zip**
JPG	0	1	1
GIF	0	0	
BMP	5	15	15
HTML	81	89	89
PS(postscript)	57	64	63
본문파일	68	71	70
PDF(portable document format	52	70	70

compress의 사용

compress는 UNIX의 초기의 압축편의프로그람들중의 하나로서 압축속도가 빠르고 사용도 간단하다.

```
compress ux2nd22                원래파일을 변경시켰다
```

이 지령은 파일 Ux2nd22.z를 산출하며 원래파일을 제거한다. uncompress로 압축파일을 풀수 있다.

```
uncompress ux2nd22.Z
```

이 지령은 원래파일을 보존한다. compress지령은 또한 여러개 파일들을 압축하는데 현재등록부에 있는 모든 파일들을 압축하기 위해서 compress *지령을 사용할수 있다. compress지령으로 압축한 파일을 보기 위하여 zcat지령을 사용할수 있는데 zcat지령은 압축된 파일을 풀어서 그 내용을 말단에 표시한다.

주해

compress, uncompress, zcat가 다 같은 프로그람(3개의 련결을 가지는)이라는것은 놀라운 일이다. 련결에 대해서는 다음장에서 론의된다.

gzip의 사용

UNIX체계에서 가장 일반적으로 쓰이고 있는 압축프로그람인 GNU gzip는 Linux에서도 쓸수 있다. compress와 비교하면 속도가 좀 느리기는 하지만 매우 효률적이다. gzip로 압축된 파일은 확장자가 .gz이다. 그 압축파일은 gunzip지령으로 푼다.

```
gzip sales.dbf                sales.dbf.gz파일이 생긴다
gunzip sales.dbf.gz           sales.dbf가 다시 생기게 한다
```

gunzip지령은 compress로 압축된 파일도 풀수 있는 만능지령이다. gzip는 오늘날 인터네트상에서

널리 리용되고 있는 표준압축편의프로그람이다.

zip의 사용

대부분의 UNIX체계들(Linux도)은 필 카쯔(Phil Katz)의 PKZIP프로그람 즉 zip도 제공하고 있다. 이 프로그람은 그것이 여러 파일들을 하나의 **보존파일**(archive)에 압축한다는 점에서 다른 압축편의프로그람들과 구별된다.

```
zip fin *.html          fin.zip파일이 생긴다
```

이 지령은 모든 HTML파일들을 하나의 파일인 fin.zip에 압축한다. unzip지령은 압축을 푸는데 리용된다. zip지령은 또한 -r선택항목을 리용하여 전체 등록부나무를 재귀적으로 압축할수 있다.

```
zip -r zipdir docs              zipdir.zip로 docs등록부구조를 압축한다
```

compress와 gzip지령들은 tar와 같은 외부응용프로그람(22.9.4)의 도움으로만 우와 같은 기능을 수행할수 있다. zip지령은 compress보다 더 나으며 gzip지령과 매우 류사하다.

인터네트상에서 압축된 파일

인터네트상에서 흔히 닉명ftp싸이트(11.7.1)들을 내리적재할 때 압축파일들을 보게 될것이다. 이 파일들은 모두가 한개의 압축파일에 여러개의 파일들을 포함하고 있다. 이 파일은 우에서 취급된 확장자들을 가지고 있으며 또한 여러개의 확장자들을 가질수 있다. 압축된 파일의 확장자들을 보고 압축지령들의 형을 추측할수 있다. 일반적으로 다음과 같은 확장자들이 있다.

- zip-zip로 압축된 파일
- tar-UNIX에서 tar지령에 의하여(여기서는 압축은 하지 않는다.) 한개의 파일로 보존된 파일
- gz-gzip로 압축된 파일
- tar.Z-tar에 의하여 하나의 파일로 보존된 다음에 compress지령으로 압축된 파일
- tar.gz-gzip지령으로 압축된 파일

tar.gz 또는 tar.Z파일은 두 단계로 만들어 진다. 즉 첫 단계에서는 tar지령으로 여러개의 파일들을 하나의 보존파일로 묶고 두번째 단계에서는 gzip 또는 compress지령으로 그 보존파일자체를 압축한다. 이것을 진행하는 방법은 22.9.4에서 취급한다.

Linux

gzip, gunzip, zcat지령은 사실상 3개의 련결(이름)을 가지는 하나의 파일이다. Linux는 zcmp, zgrep, zmore, zdiff와 같이 "z"접두사가 있는 많은 지령들을 가지고 있다. "z"접두사가 없는 지령에 대해서는 다음장들에서 론의된다.

6.20 결론

이 장에서 취급된 모든 지령들은 파일 또는 등록부이름들을 인수로 사용한다. 일부 지령들은(cat와 lp 그리고 lpr, compress와 gzip) 또한 입력을 진행하는 다른 방법도 제공한다. 더우기 cat의 출력은 항상 말단으로 나갈 필요는 없다. 많은 지령들이 때때로 건반이나 다른 프로그람으로부터 입력될수 있다. 마찬가지로 지령의 출력도 어떤 파일에로 나갈수 있으며 다른 지령에 입력으로 봉사될수 있다.

이 장에서는 이러한 방법들을 취급하지 않고 이 지령들의 일부가 어떻게 일반적으로 사용되는가 하는것을 주었다. 그러므로 화면에서 모든 지령의 출력을 보지 않아도 된다. 그리고 출력에 어떤 려파를 실시하거나 그것을 파일에 보관하는 방법에 대해서는 제8장에서 취급한다.

요 약

UNIX에서 C프로그람과 UNIX지령, 인쇄기, 테프구동기 그리고 체계의 기억기와 같은 모든것들은 파일로 취급된다. 이 파일들은 3가지 형태 즉 보통파일, 등록부파일, 장치파일형태로 존재한다.

보통파일은 자료를 포함하고 있는데 가장 일반적인 형태는 본문이다.

등록부는 파일들을 포함하며 등록부파일은 그것들의 이름을 포함한다.

장치파일은 디스크상의 공간을 차지하지 않으며 거기에 지적된 자료가 실제적인 물리적장치를 가리키게 한다.

파일은 자기 속성뿐아니라 파일끝표식도 포함하지 않는다.

파일이름은 255개 문자들로 제한되며 /을 제외한 모든 문자를 실지로 사용할수 있다.

그리고 여러개의 확장자와 점을 쓸수 있으며 대소문자를 구별한다.

파일체계는 거꾸로 된 나무와 류사한 계층구조이며 제일 꼭대기등록부를 뿌리라고 부른다. 등록부들과 파일들은 어미-새끼관계를 가지며 여러 파일의 어미는 등록부로 된다.

사용자들이 리용하는 지령들은 /bin등록부와 /usr/bin등록부안에 배치된다. 관리자지령들은 /sbin과 /usr/sbin안에 있으며 그의 구성파일들은 /etc등록부안에 있다. /dec에는 장치파일들이 들어 있다.

사용자들은 /tmp에 림시파일을 만들며 통지문들과 인쇄일감들을 /var에 줄 지워 세운다.

pwd지령은 현재등록부를 지정해 주며 cd지령은 그것을 변화시키는데 사용된다. cd지령이 인수없이 사용될 때 이 지령은 사용자의 가입등록부인 홈등록부로 절환한다.

경로이름은 사선으로 분리된 등록부와 파일이름들의 렬이다. 절대경로이름은 뿌리에 대한 그 파일의 위치를 나타낸다.

두 파일이 같은 이름으로 존재할 때 이것들은 서로 다른 절대경로이름을 가져야 한다.

파일들은 또한 상대경로이름으로 표현될수 있는데 이것은 현재등록부와 관계되는 상대적인 파일의 위치를 보여 준다.

일반적으로 기호 .과 ..기호들은 각각 현재등록부와 어미등록부를 표현하기 위하여 리용된다.

mkdir와 rmdir는 등록부를 만들거나 제거하기 위하여 사용된다.

한번의 mkdir와 rmdir지령실행으로 여러개의 등록부들을 만들거나 제거할수 있다. 만일 등록부가 텅 비지 않았다면 rmdir지령으로 등록부를 제거할수 없다.

cp지령으로 여러개의 파일들을 복사하고 rm지령으로 그것들을 제거하며 mv지령으로 그것들의 이름을 변경할수 있다.

cp와 mv는 존재하고 있는 파일들을 덧쓰기하며 이 세 지령들은 모두가 대화형식(i)으로 리용된다. cp와 rm은 전체 등록부구조에서 재귀적(r)으로 작업할수 있다. mv는 또한 두개의 등록부이름들을 변경

시킬수 있다.

rm -rf*지령은 흔히 누구나 다 오유를 범할수 있는 가장 위험한 지령이다.

cat지령은 한개 혹은 그이상의 파일을 표시할수는 없지만 한개 파일을 만든다. 건반으로부터 입력된 cat지령의 입력은 파일의 끝문자임을 나타내는 [ctrl-d]건으로 끝난다.

file지령은 일반적인 세 부류외에 다른 형의 파일을 보여 준다. 이 지령으로 tar보존파일들, 화상들, HTML파일들과 이 지령으로 압축한 파일들을 분간할수 있다.

lp지령은 파일을 인쇄하며 여러개의 복사물(-n)들을 인쇄하는데 리용될수 있다. 이 지령으로 또한 인쇄가 끝난후에 사용자에게 통보문을 통지할수 있다. cancel지령으로 제출된 어떤 일감을 취소할수 있다. Linux에서는 인쇄지령이 lpr이며 lprm지령은 일감을 취소하기 위해서 사용된다.

df와 du지령들은 디스크공간을 관리하기 위한 지령이다. df지령은 매개 파일체계의 빈 공간을 보여주며 du지령은 매개 파일 또는 등록부의 상세한 사용공간을 보여 준다. 이 지령들은 다 체계관리자에 의해 정기적으로 사용된다.

압축편의프로그람 compress, gzip, zip들은 파일들을 압축하는데 리용된다. gzip지령은 매우 일반적인 지령으로서 인터네트상에서 광범히 리용된다. zip지령은 파일묶음을 하나의 보존파일로 압축할수 있다. 압축된 파일들은 uncompress, gunzip와 unzip지령으로 풀수 있다.

zcat지령은 gzip와 compress로 압축된 파일들을 현시한다.

시험문제

1. 모든 UNIX체계에 있는 /dev/mem파일은 무엇을 나타내는가? 이 파일이 표시하는 내용은 무엇인가
2. UNIX파일이름의 길이는 몇개 문자가 될수 있는가?
3. 같은 등록부에 note와 Note파일들이 있을수 있는가?
4. 어느 등록부가 어미등록부를 가지지 못하는가?
5. 한개, 두개의 점을 포함하고 있는 파일을 만들어 보시오. 무엇을 알수 있는가?
6. 어느 문자가 [Enter]건을 발생시키는가?
7. cat/bar/bar/bar/bar라는 지령이 문법에 맞는가?
8. cd지령이 인수없이 사용될 때 무엇을 진행하는가?
9. cd cd/mkdir/rmdir지령을 사용할수 있는가?
10. ls..지령이 실행되는가?
11. 절대경로이름을 사용하여 /sbin등록부에 있는 지령을 실행시킬수 있는가?
12. share/man/catl과 같은 등록부구조를 만드는 가장 빠른 방법을 찾기 위해 UNIX문서를 들여다 보시오.
13. 자기가 위치하고 있는 등록부를 제거할수 있는가?
14. 등록부구조 bar1을 bar2로 어떻게 복사하겠는가?
15. 만일 bar1과 bar2가 등록부라면 mv bar1 bar2지령이 실행되는가?
16. 파일이 인쇄할수 없는 문자를 포함하고 있을 때 그것들을 어떻게 볼수 있는가?

17. /dev등록부안에 있는 모든 파일에 file지령을 사용하시오. 2개 부류로 묶어 놓을수 있는가?
18. 확장자가 .gz인 파일은 무슨 부류에 속하는가?

련습문제

1. 여러개의 점 실례로 5개의 점으로 된 파일을 만들수 있는가?
2. 파일이름에 사용될수 없는 문자는 어떤것인가? 파일이름에 공백이 사용될수 있는가?
3. 파일이름에 사용되지 말아야 할 문자는 어떤것인가?
4. 파일이름의 시작에 이음표를 쓸수 없는 두가지 리유를 이야기하시오.
5. 등록부파일은 무엇을 포함하는가?
6. 등록부파일의 내용을 변화시킬수 있는가?
7. charlie의 홈등록부는 /usr/charlie이며 /usr/charlie/html경로에 관계되는 많은 스크립트들이 작성되였다. 체계관리자는 현재홈등록부를 /home/charlie로 변경하였다는것을 charlie에게 알려 주고 그의 모든 스크립트들을 변경시키도록 조언을 주었다. 이 문제를 어떻게 해결할수 있는가?
8. /usr/spool/lp/admins 등록부로부터 /usr/spool/mail등록부에로 어떻게 이동할수 있는가?
9. 우리는 echo가 쉘내부지령이라는것을 알고 있다. 그러면 /bin등록부에 있는 같은 이름을 가진 지령을 어떻게 리용하겠는가?
10. 자기의 홈등록부를 알기 위한 3가지 방법을 말해 보시오.
11. 만일 mkdi foo지령이 등록부를 만들지 못한다면 그 리유는 무엇인가?
12. rmdir bar지령이 언제 실패되는가? bar등록부를 어떻게 제거하겠는가?
13. 언제 update.sh대신에 ./update.sh지령을 사용해야 하는가 ?
14. 체계관리자의 지령들과 구성파일들은 어디에 보관되여 있는가?
15. cd지령과 cd $HOME지령의 차이점은 무엇인가?
16. 현재등록부 bar1로부터 어미등록부 bar2로 어떻게 모든 등록부와 파일들을 복사하겠는가?
17. hosts파일이 현재등록부안에 존재하고 있어도 cp hosts badcup/hosts bak지령이 실행되지 못하는 리유는 무엇인가?
18. rm *지령대신에 무슨 지령을 사용하는것이 더 안전한가?
19. rm -fi *지령은 무슨 지령인가?
20. bar2가 이미 존재하거나 존재하지 않는 경우 mv bar1 bar2지령을 실행하면 어떻게 되는가?
21. cat foo foo foo는 무엇을 현시하는가?
22. System V상에서 이름이 mainp인 인쇄기에 /etc/passwd의 복사물을 3개 인쇄하시오.
23. 먼 곳의 기계에 인쇄일감을 어떻게 배포하며 일감이 인쇄된것을 어떻게 확정할수 있는가?
24. df와 du지령은 어떻게 다른가?
25. 현재등록부나무의 전체 디스크사용공간을 어떻게 알아 내는가?
26. 어떤 사람에게 보내야 할 등록부구조가 있는데 어느 지령이 이것을 실현하기 위한 가장 적합한 지령이며 어떻게 해야 하는가?
27. 확장자가 .dif인 여러개의 화상파일들이 있는데 이 파일들을 압축하기 위해서는 어느 지령을 리용해야 하는가?

제 7 장. 파일속성

제6장에서는 아무러한 문제없이 등록부들을 만들고 파일체계를 조종하며 파일들을 복사하고 이동하고 삭제하였다. 그러나 실지에 있어서는 파일 또는 등록부를 조종하는데서 문제점들이 제기될수 있다. 다른 사용자들에 의해 변경되거나 지어는 지워 질수도 있다. 여벌자료로부터 회복하는것도 불가능하다.이러한 문제점들이 생기는 원인과 이것을 어떻게 방지하며 수정할수 있는가를 알고 있어야 한다.

UNIX파일체계는 사용자들이 어떤 비밀에 해당되지 않는 파일들에 접근하게 한다. 파일도 어떤 규칙에 따라 변화될수 있는 속성들을 가지고 있다. 이러한 속성들을 현시하기 위하여 ls지령을 리용한다. 또한 이러한 속성들을 변화시키는 다른 지령들도 있다. 마지막으로 UNIX체계의 가장 만능적인 속성관리도구로서 find를 취급한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- ls지령을 리용하여 가능한 방법으로 파일들을 렬거한다(7.1).
- ls -l지령출력의 7개 마당의 의미를 파악한다(7.2).
- 파일과 등록부허가권의 의미와 그것을 chmod로 변경시키는 방법을 배운다(7.4부터 7.6까지).
- umask설정이 어떻게 기정허가권을 결정하는가를 파악한다(7.7).
- 파일소유권에 대한 개념과 chown과 chgrp로 그것을 변경시키는 방법을 배운다(7.8, 7.9).
- ls를 리용하여 파일의 마지막변경과 접근시간을 현시하고 touch로 변경시킨다(7.10, 7.11).
- 파일체계와 파일식별자로서의 색인마디에 대하여 리해한다(7.12).
- ln으로 파일에 경련결을 만들고 ls -i를 리용하여 색인마디번호를 현시한다(7.13).
- 련결의 실제적인 응용을 배운다(7.13).
- 기호련결이 왜 경련결보다 우월한가를 리해한다(7.14).
- find로 한개 또는 그이상의 파일속성들을 비교함으로써 파일들의 위치를 알아 낸다(7.15).

7.1 파일렬거(ls)

앞부분에서 우리는 현재등록부안에 있는 일부 파일들을 현시하기 위하여 ls지령을 사용하였다. 이 지령은 실제적으로 파일의 모든 속성(우리가 이 장에서 관심을 돌려야 하는것)들을 현시한다. 많은 파일이름을 포함하고 있는 등록부에서 이 지령을 다시 실행시켜 보자.

```
$ ls
08_packets.html          처음에 수자
TOC.sh         다음에 대문자
calendar                 그다음 소문자의 순서로
cptodos.sh
dept.lst
emp.lst
```

```
helpdir
progs
usdsk06x
usdsk07x
usdsk08x
```

여기서 보여 주는것은 ASCII순서맞추기렬로 배렬된 파일이름들의 목록이며 한행에 파일이름을 한개씩 보여 준다. 이 렬은 다음의 순서로 되여 있다.

공백(공간과 타브)
수자
대문자
소문자

목록은 또한 적당한 선택항목과 함께 ls지령을 사용한후 확인할수 있는 등록부들을 포함한다. 목록이 길면 모든 파일들을 볼수 없을것이다. 필요한 파일을 보고 싶은 경우에는 파일이름과 함께 ls지령을 사용해야 한다.

```
$ ls calendar
calendar
```

그리고 만일 /usr/bin등록부안에 perl파일이 없다면 ls지령은 다음과 같이 통지한다.

```
$ ls /usr/bin/per1
ls: /usr/bin/perl: No such file or directory
```

ls는 또한 여러개의 파일이름들과 함께 리용될수 있다. 그것은 수많은 선택항목들을 가지고 있는데 그 수에 있어서 다른 지령들과 대비도 안된다(대부분의 판본에서 20개이상). 우리는 파일 또는 등록부의 속성을 보여 주는 중요한것들을 취급한다. 이 선택항목들은 표 7-1에 요약되여 있다.

ls의 선택항목들

여러개의 렬로 출력(-x)

파일이 여러개 있을 때는 파일이름을 여러개의 렬로 현시하는것이 더 좋다. 여러개의 렬로 출력하려면 -x선택항목을 사용하시오.

```
$ ls -x
08-packets.html   TOC.sh      calendar    cptodos.sh
dept.lst          emp.lst     helpdir     progs
usdsk06x          usdsk07x    usdsk08x    ux2nd06
```

현시장치는 말단너비를 충분히 사용하도록 하며 많은 사람들이 기정적인 형식으로 지령들을 전용화하는데 편리하다(즉 선택항목없이 리용될 때). 후에 별명(17.4)과 쉘함수(19.10)들을 배운 다음에는 그것들을 아주 쉽게 할수 있다.

등록부들과 실행파일들을 식별하기(-F)

지금까지 본 ls지령의 출력은 단지 파일이름들만 보여 주었다. 만일 어떤 곳에 등록부파일이 있다면

표 7-1. ls의 선택항목들

선택항목	의 미
-x	출력을 여러 렬로 현시
-F	실행파일은 *로, 등록부들은 /으로, 기호련결은 @로 표식한다
-a	. 과 ..이 들어 있는 모든 파일들과 하나의 점으로 시작되는 파일들을 보여 준다
-R	모든 보조등록부안에 있는 파일들을 재귀적으로 보여 준다
-L	기호련결로 지적된 파일들을 보여 준다
-d	등록부의 목록을 보여 준다
-l	ASCII순서맞추기렬로 파일의 7개 속성들을 보여 준다
-n	파일의 이름대신 수자로 된 사용자ID와 그룹ID를 현시한다
-t	마지막으로 변경된 시간순서로 파일들을 정렬한다
-lt	마지막으로 변경된 시간순서로 정렬된 목록을 현시한다
-u	마지막으로 호출된 시간순서로 정렬한다
-lu	목록안에 마지막으로 호출된 시간을 현시하는데 ASCII순서맞추기렬로 정렬한다(Linux에서는 호출된 시간순서로 정렬한다)
-lut	마지막으로 호출된 시간순서로 정렬하고 현시한다(Linux에서 -lu지령과 같다)
-I	색인마디번호를 보여 준다
-c	색인마디가 마지막으로 변경된 시간순서로 정렬한다
-s	512KB블로크단위로 파일크기를 보여 준다(Linux에서는 1024B)
-r	파일들을 반대순서로 정렬한다(기정적으로는 ASCII순서맞추기렬에 의하여)

그것들이 얼마나 많은지 모른다. 등록부와 실행파일들을 구별하려면 -F선택항목을 리용하여야 한다. -x선택항목과 결합하면 출력을 여러개의 렬로 보여 준다.

```
$ ls -Fx
08-packets.html    TOC.sh*      calendar*    cptodos.sh*
dept.lst           emp.lst      helpdir/     progs/
usdsk06x           usdsk07x     usdsk08x     ux2nd06
```

일부 파일이름들에 2개의 꼬리표가 붙어 있는데 *는 파일이 실행코드를 포함하고 있다는것을 지적하며 /은 등록부이라는것을 의미한다. 따라서 현재등록부에서 2개의 보조등록부 즉 helpdir와 progs등록부를 구별할수 있다.

숨겨 진 파일들을 보기(-a)

기정적으로 ls지령은 등록부안에 있는 모든 파일들을 보여 주지 않는다. 매 등록부에는 숨겨 진 파일(점으로 시작되는 파일)들이 있을수 있다. 특별히 홈등록부에 있는데 일반적으로는 보이지 않는다. -a(all)선택항목은 다음과 같이 숨겨 진 파일들을 보여 준다.

```
$ ls -axF
./          ../              .cshrc         .emacs
.exrc       .fetchmailrc     .netscape/     .profile
.rhosts     .sh_history      .xinitrc       08_packets.html*
TOC.sh      calendar*
....
```

일반적으로 이 《점》파일들은 대응하는 지령들의 동작을 결정한다. emacs는 호출될 때 .emacs를 읽는데 netscape는 기동시에 .netscape등록부에 있는 많은 파일들을 읽는다. X Window체계는 .xinitrc안에 있는 명령들을 실행한다.

.cshrc와 .profile파일들에 대해서는 특별한 언급이 필요하다. 이 파일들은 사용자가 가입할 때 수행되는 명령묶음을 포함하고 있다. 개념적으로는 Windows에 있는 AUTOEXEC.BAT파일과 류사한데 후에 이 파일들에 대하여 더 구체적으로 취급한다(17.9). 일부 장들에서 여러가지 형의 숨겨 진 파일들의 의미를 론의할것이다.

첫 두개 파일(.과 ..)들은 현재등록부와 어미등록부(6.8)를 나타내기 위해서 사용된 기호들을 가진 특수한 등록부들이다. 이 기호들은 여기서 같은 의미를 가지는데 보조등록부를 만들 때마다 보이지 않는 이 등록부들은 체계에 의하여 자동적으로 만들어 진다. 그 등록부들은 제거할수 없을뿐아니라 또한 쓰기도 할수 없다. 이것들도 파일체계에 함께 포함되여 있다.

주해

점으로 시작되는 모든 파일이름들은 오직 ls지령이 -a선택항목과 함께 사용될 때만 현시된다. 점(.)등록부는 현재등록부임을 나타내며 두개의 점(..)은 어미등록부임을 의미한다. 또한 ls지령에 인수로서 -a선택항목을 지적함으로써 숨겨 진 파일을 볼수 있다.

등록부의 내용보기

여기서는 선택항목을 론의하지 않고 등록부의 내용을 보는 방법을 취급한다. 앞실례(ls calendar)에서는 calendar파일이 존재하는가를 검사하기 위해서 ls지령에 보통파일이름을 지정하였다. 그러나 대신에 progs등록부이름을 지정하면 다르게 현시될것이다.

```
$ ls -x progs
array.pl      cent2fah.pl      n2words.pl      name.pl
```

등록부의 내용이 현시되는데 몇개의 perl파일들로 구성되여 있다. 그 출력은 내용이 현시된 등록부를 지적하지 못한다.

주해

만일 ls지령에 인수로서 한개의 파일이름이 사용될 때 이 지령이 어떤 파일목록을 현시한다면 인수로 지정된 파일이름이 실제로는 등록부라는것을 알수 있다. 그때 ls는 등록부의 내용을 보여 준다. -d선택항목은 이 동작을 막는다. 등록부속성을 론의할 때 이 선택항목을 취급한다.

재귀적으로 목록을 보여주기(-R)

-R(recursive)선택항목은 등록부나무안에 있는 모든 파일들과 보조등록부들을 보여 준다. 그 등록부나무의 현시는 보조등록부가 없을 때까지 재귀적으로 수행된다.

```
$ ls -xR
08-packets.html     TOC.sh          calendar         cptodos.sh
dept.lst            emp.lst         helpdir          progs
usdsk06x            usdsk07x        usdsk08x         ux2nd06
./helpdir:
forms.hlp           graphics.hlp    reports.hlp
./progs:
```

```
array.p        l     cent2fah.pl    n2words.pl        name.pl
```

이 목록은 파일들을 3개 부분으로 보여 준다. 즉 홈등록부안에 있는 파일들과 helpdir와 progs보조등록부안에 있는 파일들을 보여 준다. 이미전에 배운 이름짓기규칙에 의하여 이름이 지어 진 보조등록부가 나타난다. 즉 목록에서 ./helpdir는 현재등록부밑에 있다. 만일 이 현재등록부가 /home/romeo이라면 helpdir등록부의 절대경로이름은 /home/romeo/helpdir로 확장된다.

주해 어떤 선택항목들은 현시순서를 반대로 하기 위해 -r선택항목과 함께 사용될수 있다.-r선택항목은 ASCII렬을 반전시켜 파일들을 보여 준다.

7.2 파일속성렬거(ls -l)

ls지령의 -l(long)선택항목은 파일의 기본속성 즉 파일허가권, 파일크기, 소유권과 같은 세부정보들을 보여 준다. UNIX전문용어에서는 흔히 출력을 **렬거**(listing)라고 하는데 대표적인 렬거를 그림 7-1에 보여 준다.

```
$ ls -l
total 35
-rw-r--r--   1  romeo    metal  19514   May 10 13:45   chap01
-rw-r--r--   1  romeo    metal   4174   May 10 15:01   chap02
-rw-rw-rw-   1  romeo    metal     84   Feb 12 12:30   dept.lst
-rw-r--r--   3  juliet   metal   9156   Mar 12  1999   genie.sh
drwxr-xr-x   2  romeo    metal     64   May  9 10:31   helpdir
drwxr-xr-x   2  romeo    metal    320   May  9 09:57   progs
```


그림 7-1. ls -l지령에 의한 파일들의 긴 렬거

목록의 맨앞에 total 35라는 단어가 있는데 이것은 파일들이 디스크에서 총 35개 블로크를 차지한다는것을 보여 준다(매개 블로크는 512 또는 1024byte를 포함한다). -l선택항목은 더 많은 속성들을 현시하며 정렬순서를 반대로 하기 위하여 다른 속성들과 함께 조합될수 있다. 그림에서 보여 준 7개 마당들의 의미를 론의하자.

형과 허가권

첫번째 렬은 매개 파일과 관련된 형과 **허가권**(permission)을 보여 준다. 이 렬에서 첫번째 문자는 첫 4개 파일들에 대해서 -로 되여 있는데 그것은 그 파일이 보통파일이라는것을 의미한다. 그러나 helpdir와 progs등록부에 대해서는 그 위치에 d가 있으므로 보통파일이 아니다.

다음에는 r, w, x와 -로 된 문자렬이 보인다. UNIX체계에서 파일은 3가지 즉 읽기, 쓰기, 실행의 허가권을 가질수 있다. 이 허가권들을 해석하는 방법과 변경하는 방법은 7.5에서 취급한다.

련결

두번째 렬은 파일과 련관된 **련결**(link)들의 개수를 지적한다. 이것은 사실상 그 파일체계에 의해 유지되는 이름들의 개수이다. UNIX는 설사 디스크상에 한개 파일이 있다 할지라도 가지고 싶은것만큼 많은 이름들을 가지게 한다. 여기서 genie.sh파일은 3개의 련결을 가지는데 다른 련결들은 서로 다른 등록부에 있을수 있다. 이 속성은 7.13에서 취급한다.

주해

련결수가 1이상이라는것은 그 파일이 하나이상의 이름을 가진다는것을 의미한다. 그러나 파일의 복사판이 여러개 있다는것을 의미하지는 않는다.

소유권과 그룹소유권

어떤 파일을 만들 때 만든 사람은 자동적으로 그 파일의 **소유자**(owner)가 된다. 세번째 렬은 genie.sh파일을 제외하고는 모든 파일의 소유자가 romeo라는것을 보여 준다. 소유자는 그 파일의 내용, 허가권, 소유권을 조작할수 있는 모든 권한 즉 뿌리사용자를 제외한 다른 사용자들에게는 불가능한 권한(privilege)을 가지고 있다. 여기서 romeo는 juliet와 뿌리사용자에 의해서도 변경될수 있는 genie.sh파일을 제외한 모든 파일들의 속성을 변경시킬수 있다.

사용자등록자리를 열 때 체계관리자는 어떤 그룹에 그 사용자를 할당한다. 네번째 렬은 그 파일의 **그룹소유자**(group owner)를 보여 준다. 파일을 소유하고 있는 사용자들의 그룹에 대한 개념은 그룹성원들이 때때로 같은 파일에서 작업할것을 요구하기때문에 중요성을 띤다. 일반적으로 **그룹**(group)은 그 소유자와 마찬가지로 다른 사람들과 완전히 다른 권한모임을 가진다. 소유권과 그룹소유권은 7.8에서 취급된다.

크기

다섯번째 렬은 파일의 크기를 바이트단위로 보여 준다. 이것은 사실상 문자수이지 파일이 차지하고 있는 디스크공간이 아니다. 디스크상에 리용된 공간은 파일이 1024byte(일반적으로)의 블로크단위로 디스크에 씌여 지므로 이 그림에서 보여 주는것보다 실지로는 더 크다. 다시 말해서 dept.lst파일의 크기가 84byte라고 해도 디스크상에는 1024byte를 차지한다.

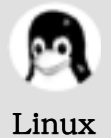
두개의 등록부들은 더 작은 파일크기를 보여 준다. 등록부는 매 파일에 대해서 식별번호(색인마디)와 함께 파일이름의 목록을 가지고 있다. 따라서 등록부파일의 크기는 이 목록의 크기 즉 그 파일들자신의 크기에 의존한다.

마지막변경시간

다음 3개의 렬은 파일의 마지막변경시간 즉 제일 가까운 초로 보관된 시간도장(time stamp)을 지적한다. 어떤 파일의 내용이 아무런 방법으로 변화되였으면 그 파일은 변경되는것으로 간주된다. 만일 파일이 마지막으로 변경된 날자로부터 1년이상 지났다면 년도가 표시된다. 실례로 genie.sh파일을 들수 있다.

때때로 파일의 변경시간에 기초하여 결심하는 자동적인 도구들을 실행하는것이 필요할것이다. 이 렬은 ls가 어떤 선택항목들과 함께 사용될 때 다른 두개의 시간도장을 보여 준다. 시간도장은 7.10에서 취급된다.

파일이름

마지막렬은 자모순서로 배렬된 파일이름을 현시한다. 6.2에서 이미 UNIX파일이름이 255문자까지 될수 있다는것을 취급하였다. 또한 모든 파일이름들사이에 수자, 이음표(-), 밑선(_), 점(.)을 끼워 놓을수 있다.

Linux

Linux는 파일의 마지막변경시간이 6달 이전의것이라면 목록의 출력에 그 년도를 보여 준다. 또한 1024byte블로크단위로 보여 준다. 실례로 ls -l출력의 첫행(total단어가 현시되는 행)은 1024인 블로크크기를 사용한다.

7.3 등록부속성렬거(ls -d)

ls는 등록부이름과 함께 사용되면 등록부안에 있는 파일들을 보여 준다. 내용보다도 등록부속성을 보고 싶다면 -d선택항목을 사용해야 한다.

```
$ ls -ld helpdir progs
drwxr-xr-x      2 romeo  metal          48 Jan 30 14:21 helpdir
drwxr-xr-x      2 romeo  metal          96 Mar 12 14:50 progs
```

-l선택항목은 여기서 등록부목록이 보통파일과 다르다는것을 보여 주기 위하여 결합되였다. 등록부들은 첫번째 렬의 첫번째 문자에 의해서 쉽게 구별되는데 그 첫번째 문자는 항상 d이다. 보통파일인 경우에는 이 위치에 항상 -(이음표)를 보여 주며 장치파일들인 경우에는 b 혹은 c로 된다. 등록부속성은 7.6에서 취급된다.

주해

만일 등록부안에 포함된 파일보다도 등록부속성을 보고 싶다면 등록부이름과 함께 ls -d를 사용해야 한다. ls -d지령을 사용한다면 현재등록부안에 있는 보조등록부들을 보여 주지 않는다. ls는 등록부들만 보여 주는 선택항목을 가지고 있지 않다.

7.4 파일허가권

UNIX는 간단한것 같지만 파일들에 **허가권**(permission)을 할당하는 잘 정의된 체계이다. 그것들을 리해하자면 소유권과 그룹소유권의 의미를 잘 리해하여야 한다. 등록부안에 있는 일부 파일들을 보기 위하여 ls -l지령을 다시한번 주자.

```
$ ls -l chap02 dept.lst dateval.sh
-rwxr-xr--   1 romeo    metal      20500 May 10 19:21 chap02
-rwxr-xr-x   1 romeo    metal        890 Jan 29 23:17 dateval.sh
-rw-rw-rw-   1 romeo    metal         84 Feb 12 12:30 dept.lst
```

파일허가권을 보여 주는 첫번째 렬을 잘 보시오. 이 허가권들은 3개 파일들에 대해서 다르다. UNIX는 어떤 파일에 대한 접근렬을 결정하는 3층으로 된 파일보호체계를 따른다. 매층은 하나의 **부류**(category)를 표현하는데 허가권의 3가지 형을 표현하기 위하여 r, w, x문자렬을 리용한다.

r는 읽기허가권을 지적하는데 cat가 파일을 현시할것이라는것을 의미한다. w는 쓰기허가권을 지적한다. 즉 편집기로 파일을 편집할수 있다. x는 실행허가권을 지적한다. 즉 그 파일은 어떤 프로그람으로서 실행될수 있다. r, w, x의 전반적인 문자렬의 의미를 리해하려면 그림 7-2에서 보여 주는것처럼 3가지 묶음으로 갈라 놓고 리해하여야 한다.

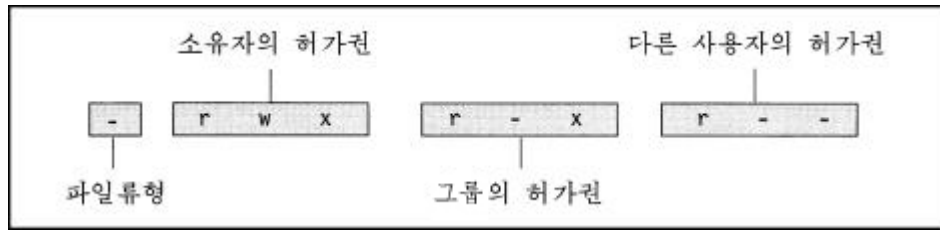



그림 7-2. 파일허가권문자렬의 구조

chap02파일의 허가권을 나타내는 이 그림에서 첫번째 묶음(rwx)은 3개의 허가권을 다 가진다. 이 파일은 그 파일소유자에 의해 읽기할수 있고, 쓰기할수 있으며, 실행할수 있다. 그렇다면 소유자는 누구인가? 목록의 3번째 렬은 소유자가 romeo라는것을 보여 준다. 사용자는 이러한 권한들을 리용하려면 romeo라는 사용자이름으로 가입하여야 한다.

두번째 묶음(r-x)은 가운데위치에 이음표를 가지고 있는데 파일의 그룹소유자에게 적용된다. 그룹소유자는 metal인데 그 그룹에 속한 모든 사용자들은 오직 읽기와 실행만을 할수 있는 허가권을 가지고 있다.

세번째 묶음(r--)에는 쓰기, 실행비트들이 없다. 이 허가권모임은 다른 사용자 즉 소유자가 romeo도 아니고 metal그룹에도 속하지 않는 사람들에게 적용되는것이다.

그룹, **다른 사용자**(others)라는 범주를 흔히 **객관**(world)이라고 부르기도 한다. 파일이 《객관쓰기가능(world-writable)》하다는것은 그룹과 다른 사용자들에 대한 허가권문자렬부분에 W가 있다는것(r-rw-rw-와 같이)을 의미한다.

주해 romeo가 metal그룹의 한 성원일지라도 그룹허가권은 romeo에게 적용되지 않는다. 소유자는 그룹소유자의 허가권을 무시하는 자기의 허가권모임을 가진다. 그러나 romeo가 파일의 소유권을 포기할 때 그룹허가권은 그에게 적용된다.

이 허가권들은 모두 변경시킬수 있지만 주의해야 한다. 만일 모든 부류의 사용자들이 읽기, 쓰기, 실행할수 있다면 허가권마당은 다음과 같이 될것이다.

```
rwxrwxrwx              모든 사람들에게 모든 허가권이 적용
```

모든 파일들에 대해서 읽기,쓰기,실행을 할수 없어야 한다. 그렇지 않으면 안전한 체계를 가질수 없다. UNIX체계는 기정적으로는 이러한 허가권들을 가진 파일을 만들지 못하게 되여 있다.

```
---------              그 누구에게도 적용되는 허가권이 없다
```

만일 chap01이 우에서 보여 준것처럼 허가권들을 가지지 않았다면 그것들을 아주 쉽게 검사할수 있다.

```
$ cat chap01           파일을 읽을수 없다
cat: chap01: Permission denied
$ echo "Come to the Web" > chap01
chap01: cannot create          파일에 쓸수 없다
$ chap01
chap01: cannot execute         파일을 실행시킬수 없다
```

UNIX는 세 부류의 사용자들 즉 소유자, 그룹, 다른 사용자들에게 서로 다른 허가권을 설정할수 있게 하는 잘 정의된 보호기구를 제공한다. 이것은 3가지 형태의 모든 파일들에 적용할수 있는데 그것들을 리해하는것이 중요하다.

7.5 파일허가권의 변경(chmod)

UNIX의 기정보안기능은 파일소유자가 아닌 사람들이 파일에 쓰는것을 방지한다. 일반적으로 모든 사용자들은 기정적으로 읽기접근을 가지지만 체계에 따라 서로 다를수 있다. 기정적인 파일허가권들은 small파일을 만들어 쉽게 알아 볼수 있다.

```
$ echo hello web > small
$ ls -l small
-rw-r--r--    1 romeo      metal      10    May 10 20:30 small
```

기정적으로 이 파일에 대해서는 소유자조차도 실행허가권을 가지지 못한다는것을 알수 있다. 그러면 이러한 파일을 어떻게 실행시키는가? chmod(change mode)로 그 파일의 허가권을 변경시킨다. 오직 그 파일의 소유자만이 이 지령을 사용할수 있다. chmod는 세가지 부류의 모든 사용자들(소유자, 그룹, 다른 사용자들)에 대한 파일허가권(읽기,쓰기,실행)을 설정한다. 그리고 문법은 다음과 같다.

```
chmod category operation permission file(s)
```

chmod는 약간 류다른 지령으로서 일반적인 지령구조와는 좀 다르다. 인수를 가진 chmod지령의 구조를 그림 7-3에서 보여 준다. 그 지령에 리용되는 식은 3가지 구성요소를 포함하고 있다.

- 사용자부류(소유자, 그룹소유자 또는 다른 사용자들)
- 수행되여야 할 연산(허가권을 할당하거나 제거)
- 허가권형태(읽기, 쓰기, 실행)

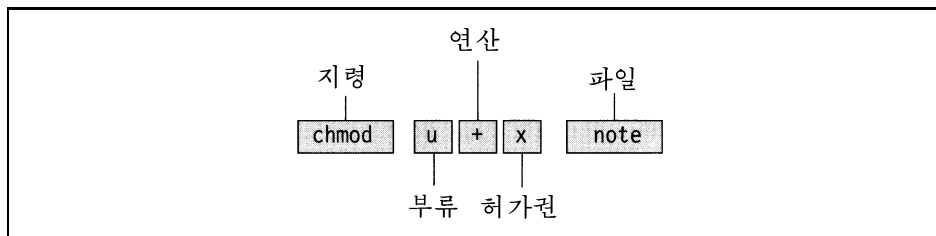



그림 7-3. chmod지령의 구조

매 구성요소들에 적당한 략어를 사용함으로써 간결한 문자렬을 구성할수 있으며 그다음에 chmod에 인수로 그 문자렬을 사용한다. 이 세가지 구성요소들에 리용되는 생략문자를 표 7-2에서 보여 준다.

표 7-2. chmod에 리용되는 생략기호

부 류	연 산	허가권
u—User	+ — 허가권할당	r—읽기허가권
g—Group	- — 허가권해제	w—쓰기허가권
o—Others	= — 절대적인 허가권할당	x—실행허가권
a—All		

실례를 들어 보자. 다른 허가권들을 변화시키지 않고 파일 small의 소유자에게 실행가능한 허가권을 할당한다. 표를 주의 깊게 보면 식은 u+x로 될것을 요구하고 있다.

```
$ chmod u+x small                소유자를 위한 실행허가권
$ ls -l small
-rwxr--r--1    romeo       metal        10    May 10 20:30 small
```

지령은 사용자(u)에게 실행허가권(x)을 할당(+)한다(chmod에서 사용자는 소유자로 리해하여야 한다). 소유자 romeo는 그 파일을 실행할수 있지만 다른 부류(그룹과 다른 사용자들)는 아직까지도 실행할수 없다. 모두에게 실행가능하게 하려면 그 부류에 해당한 문자를 사용해야 한다.

```
$ chmod ugo+x small ; ls -l small               모든 부류를 위한 x
-rwxr-xr-x     1 romeo     metal        10    May 10 20:30 small
```

ugo는 3개의 모든 부류 즉 사용자(user), 그룹(group), 다른 사용자(other)들에게 적용된다. chmod는 또한 ugo의 동의어로서 생략기호 a(all)를 제공한다. 그것으로서도 충분하지 못하다면 더 짧은 형태 즉 전혀 부류를 지적하지 않는 방법도 있다. 그러므로 이전의 실례를 다음의 형식으로 교체할수 있다.

```
chmod a+x small         a는 ugo를 의미한다
chmod +x small          기정적으로 a를 의미한다
```

허가권들은 -연산자에 의해 제거된다. 그룹과 다른 사용자들에 대한 읽기허가권을 제거하려면 식 go-r를 리용한다.

```
$ ls -l small
-rwxr--r--    1 romeo     metal        10    May 10 20:30 small
$ chmod go-r small ; ls -l small
-rwx------    1 romeo     metal        10    May 10 20:30 small
```

이 파일에 대한 모든 허가권들을 제거하려면 다음과 같이 하여야 한다.

```
$ chmod u-rwx small ; ls -l small
----------    1 romeo     metal         10    May 10 20:30 small
```

이 파일은 읽기, 쓰기, 실행할수도 없는데 제거는 할수 있다. 그렇게 해보자.

```
$ rm small
rm: small: override protection 0 (yes/no)? yes
```

rm은 어떤 사용자가 쓰기허가권을 가지지 못한 파일을 그 파일의 소유자가 지우려고 할 때 대화적인 방식을 제공한다. Solaris체계의 프롬프트상에서 yes로 설정하면 그 파일을 지운다. 또 다른 응답으로서 no로 설정하면 그 파일은 남겨 둔다.일부 체계들에서는 y로 한다. 만일 rm의 프롬프트가 시끄럽다면 rm -f를 사용하시오.

주의

어떤 문서화되지 않은 chmod의 기능은 모든 부류에 대한 쓰기허가권을 설정하는 +w를 제한한다. 명백하게 a+w 혹은 ugo+w를 사용해야 한다. 또한 -w는 오직 소유자에 대해서 쓰기허가권을 제거한다. 모든 사용자들에 대한 쓰기허가권을 제거하려면 a-w 혹은 ugo-w를 사용해야 한다.

7.5.1 다중식의 사용

다중식과 구분문자로서 반점을 리용하면 chmod는 요구하는대로 허가권을 설정할수 있다. 실례로 small파일(현재는 ---------)에 원래의 허가권(rw-r--r--)을 설정하려면 모든 부류에 읽기허가권을 할

당하는것과 함께 소유자에게 쓰기허가권을 할당하여야 한다.

```
$ chmod a+r,u+w small ; ls -l small
-rw-r--r--    1 romeo     metal        10    May 10 20:30 small
```

소유자로부터 쓰기허가권을 제거하고 그룹에 모든 허가를 할당하며 다른 사용자들에게 읽기와 실행허가권들을 할당하려면 다음의 3개 식을 사용하여야 한다.

```
$ chmod u-w,g+wx,o+x small ; ls -l small
-r--rwxr-x    1 romeo     metal         10    May 10 20:30 small
```

할당된 허가권들은 지금까지 상대적이였다. u+r식을 사용하면 다른 허가권들 즉 그룹과 다른 사용자들에 대한 허가권들은 제외하고 소유자에 대한 읽기허가권을 할당한다. chmod는 또한 절대적인 할당기능도 제공하는데 그에 대해서는 다음항목에서 취급하게 된다.

7.5.2 절대할당

chmod에 의한 절대할당은 =연산자로 수행된다. +나 -연산자와는 달리 이 연산자는 =와 함께 지적된 허가권들만 할당하며 다른 허가권들은 제거한다. 즉 만일 small파일의 3가지 모든 부류에 대해서 읽기허가권만을 할당하고 다른 모든 허가권들을 제거하고 싶다면 다음과 같이 할수 있다.

```
chmod g-wx,o-x small
```

혹은 다음의 방법들로 간단히 =연산자를 리용함으로써 할수 있다.

```
chmod ugo=r small
chmod a=r small
chmod =r small
```

때때로 어떤 파일의 현재허가권들이 무엇인지 몰라도 9개 허가비트들을 모두 설정하고 싶은 때가 있다. 절대할당방법은 가치가 있으며 사용하기 더 쉽다.

주해

파일의 허가권은 그 파일의 소유자(chmod가 리해한 사용자)에 의해 변경될수 있다. 한 사용자는 또 다른 사용자에 속한 파일들의 보호방식을 변화시킬수 없다. 그러나 체계관리자는 소유권에 관계없이 허가권을 가진 모든 파일속성들을 강제로 변경시킬수 있다.

7.5.3 8진표기법

파일허가권들을 표현하는 간략표기법(shorthand notation)이 있다. chmod는 또한 부류와 허가권을 둘 다 표현하는 수값인수를 가진다. 표기법은 8진수들(10을 기초로 하는 표준10진수체계와 달리 8을 기초로 하는 수)을 리용한다. 모든 허가권에는 아래에서 보여 주는것처럼 수값이 할당된다.

읽기허가권-4
쓰기허가권-2
실행허가권-1

어느 한 부류가 다중허가권을 가지면 련관된 수값이 첨부된다. 실례로 만일 소유자가 읽기와 쓰기허가를 가진다면 이 부류의 허가권들은 수 6(4+2)으로 나타난다. 이런 과정이 다른 부류들에 반복되면 소유자, 그룹, 다른 사용자들의 차례로 3개의 8진수문자를 가진다. =연산자와 같이 할당은 절대적이다.

모든 세 부류에 읽기와 쓰기허가권을 할당하는데 이 방식을 사용할수 있다. 8진수를 리용하지 않고 그 과제를 수행하자면 보통 chmod ugo+rw small을 리용한다. 그러나 다른 방식을 리용할수 있다.

```
$ chmod 666 small ; ls -l small
-rw-rw-rw-    1 romeo    metal         10    May 10 20:30 small
```

6은 읽기와 쓰기허가(4+2)권을 지적한다. 파일에 원래의 허가권들로 설정하려면 그룹과 다른 사용자들로부터 쓰기허가권(2)을 제거하는것이 필요하다.

```
$ chmod 644 small ; ls -l small
-rw-r--r--    1 romeo    metal         10    May 10 20:30 small
```

소유자에게 모든 허가권을, 그룹에는 읽기와 쓰기허가권을 그리고 다른 사용자들에게는 오직 실행허가권만을 할당하려면 다음의 지령중 어느 하나를 사용할수 있다.

```
chmod u=rwx,g=rw,o=x small
chmod 761 small          이것이 훨씬 더 간단하다
```

매 부류에 가능한 가장 큰 수는 7(4+2+1)이며 가장 낮은 수는 0이다. 그러므로 777은 모든 부류에 모든 허가권들이 할당되였다는것을 의미하는데 000은 모든 부류에 아무런 허가권도 할당되지 않았다는것을 의미한다. 000을 사용한후의 결과는 분명하지만 만일 어떤 파일이 모든 허가권을 가진다면 어떤 일이 일어 나겠는가?

이 대답은 예견하지 못한것이지만 모든 사용자들이 쓰기 가능한 이 파일은 편집될수 있고 재쓰기될수 있으며 추가될수 있다. 그러나 오직 소유자만이 그 파일을 지울수 있다. 이런 파일들은 누군가가 실지로 지우지는 않고 그 파일들로부터 매 바이트를 제거할수 있으므로 약간한 차이를 가진다. 표 7-3은 chmod에 8진수표기법을 사용한 경우와 사용하지 않은 경우를 보여 준다.

표 7-3. chmod사용

초기허가권	기호표현	8진수표현	변화된 허가권
rw-r-----	o+rw	646	rw-r--rw-
rw-r--r--	u-w,go-r	600	r--------
rwx------	go+rwx	777	rwxrwxrwx
rwxrw--wx	u-rwx,g-rw,o-wx	000	---------
---------	+r	666	r--r--r--
r--r--r--	+w	644	rw-r--r--
rw-rw-rw-	-w	466	r--rw-rw-
rw-rw-rw-	a-w	444	r--r--r--
---------	u+w,g+x,o+x	251	-w-r-x--x
rwxrwxrwx	a=r	444	r--r--r--

7.5.4 재귀연산(-R)

이때까지 chmod가 오직 한개 파일과 함께 리용되였지만 여러개의 파일과도 함께 작업할수 있다. 한개의 chmod지령을 사용하여 어떤 파일묶음에 허가권들을 똑같이 할당할수 있다.

```
chmod u+x note note1 note3
```

모든 파일들과 보조등록부들에 chmod지령을 재귀적으로 적용할수 있다. 이것은 -R(recursive)선택

항목과 함께 수행되며 인수로서 오직 등록부이름이나 메타문자 *가 필요하다.

```
chmod -R a+x progs          progs등록부의 모든 파일들에 재귀적으로 작용한다
chmod -R a+x *              현재등록부의 모든 파일들에 재귀적으로 작용한다
```

앞에서(6.12) 현재등록부안에 있는 모든 파일들을 정합하는데 *를 리용하였다. 우의 두개 chmod지령들은 모든 파일들과 보조등록부(prog등록부나 현재등록부안에 있는 파일)들을 모든 사용자들이 실행할수 있도록 만든다. 만일 홈등록부에 대해서 chmod를 사용하고 싶다면 거기에 《cd》하고 다음과 같은 지령을 사용하시오.

```
chmod -R 755              점은 현재등록부이다
```

여기에 중요한 문제점이 있다. 이제까지는 보통파일들의 허가권을 변경시켰다. 여기서 -R선택항목과 함께 사용된 이 지령은 또한 나무길(tree walk)안에 있는 모든 등록부의 허가권을 변화시킨다. 이 지령들이 등록부에 적용될 때 허가권은 무엇을 의미하겠는가? 다음절에 그 대답이 있다.

7.6 등록부허가권

등록부는 매 파일에 대한 식별번호와 함께 파일이름목록을 보관한다. 읽기허가를 가지고 있다고 해도 읽을수 없는 파일도 있으며 그것들에 쓰기보호되여 있어도 다른 사용자들에 의하여 제거될수 있다. 이것은 큰 충격을 줄수 있지만 사실이다. 즉 파일의 접근권은 그 등록부의 허가권에 의해서도 영향을 받게 된다. 우선 등록부의 기정적인 허가를 검사하자.

```
$ ls -ld progs
drwxr-xr-x    2 romeo    metal       128    Jun 18 22:41 progs
```

매 등록부는 허가권문자렬의 첫번째 렬에 d를 보여 준다. 기정적으로 모든 사용자들은 등록부에 대한 읽기, 실행접근이 허락되여 있다. 물론 등록부허가권은 그 의미가 다르다.

주해　mkdir로 등록부를 만들수 있으며 그다음에 기정적인 허가가 여기서 보여 준것과 같은가를 확인하기 위하여 그것의 허가권을 볼수 있다.

읽기허가권

등록부에 대한 읽기허가권은 ls가 그 등록부안에 보관된 파일이름의 목록을 읽을수 있다는것을 의미한다. 만일 그 읽기허가권을 제거하면 ls는 그 작업을 할수 없을것이며 표준편의프로그람들도 그 등록부정보를 읽을수 없을것이다.

```
$ chmod -r progs ; ls -ld progs
d-wx--x--x    2 romeo    metal       128    Jun 18 22:41 progs
$ ls -l progs              그 등록부안에 있는 파일들을 보려고 한다
ls: can not access directory progs: Permission denied (error 13)
```

그러나 만일 사용자가 파일이름을 기억하고 있다면 파일들을 따로따로 읽을수는 있다. ls는 그 이름을 현시하지는 않을것이다. 이것은 좋은 보안기능이다.

쓰기허가권

등록부에 대한 쓰기허가권은 그안에 파일이름을 만들거나 제거할수 있는 허가를 받았다는것을 의미한다. 읽기허가권을 보존하고 쓰기허가권을 제거한 다음 이 등록부에 어떤 파일을 하나 복사해 보시오.

```
$ chmod 555 progs; ls -ld progs
drwxr-xr-x    2 romeo    metal       128    Jun 18 22:41 progs
$ cp emp.lst progs
cp: unable to create file progs/emp.lst: Permission denied
```

이 등록부는 쓰기허가권을 가지지 못했다. 즉 그안에서 어떤 파일을 만들거나 복사할수 없다. 현재 존재하는 파일들에 재쓰기하거나 혹은 추가할수 있는가? 이 방법으로 등록부파일이 변화되는가? 이 질문에는 즉시 대답할수 있을것이다.

만일 모든 사용자들이 어떤 등록부에 쓰기할수 있도록 허락해 주면 어떤 현상이 일어 나겠는가?

```
$ chmod 777 progs ; ls -ld progs
drwxrwxrwx    2 romeo    metal       128    Jun 18 22:41 progs
```

이것은 아무때나 할수 있는데 아주 위험한것이다. 개별적인 파일들이 가질수 있는 허가권에는 관계없이 모든 사용자들이 이 등록부안에 있는 파일들을 제거할수 있다.

주해

등록부안의 소유자에 대한 쓰기허가권은 소유자가 그 등록부파일을 직접 편집할수 있다는것을 의미하지는 않는다. 그 권한은 오직 핵심부만이 가지게 된다. 만약 그렇지 않으면 어떤 사용자가 파일체계를 완전히 파괴할수 있다.

실행허가권

등록부의 실행권은 어떤 사용자가 보조등록부탐색을 위하여 그 등록부를 "통과"할수 있다는것을 의미한다. 만일 지령을

```
cat /home/romeo/progs/emp.sh
```

로 준다면 완전한 경로이름에 포함된 매 등록부에 대해서 실행허가권이 필요하다. 만일 단 하나의 등록부라도 이 허가권을 가지지 못하면 그다음 등록부의 이름을 탐색할수 없게 된다. 이것은 cd로도 그 등록부에 옮겨 갈수 없다는것을 의미한다.

```
$ chmod 666 progs ; ls -ld progs
drw-rw-rw-    2 romeo    metal       128    Jun 18 22:41 progs
$ cd progs
progs: permission denied
```

정규파일(regular file)들과 마찬가지로 등록부허가권은 체계보호가 거기에 의존되기때문에 매우 중요하다. 만일 등록부의 허가권을 변경시켰다면 그 허가권을 정확히 설정하였는가 확인하시오. 그렇지 않으면 어떤 사용자가 그것을 악용할수 있다.

주해

만일 등록부가 그룹과 다른 사용자들에 의해서 쓰기가능하다면 이 부류들에 속한 어떤 사용자가 그 등록부안에 있는 모든 파일을 지울수 있다. 누가 그 파일의 소유자인가 혹은 파일자체가 그 사용자에 대한 쓰기허가권을 가지고 있는가 없는가 하는것은 문제로 되지 않는다. 꼭 그렇게 해야할 리유가 없는한 일반적으로는 등록부를 객관쓰기가능한것으로 만들지 말아야 한다. 그 리유에 대해서는 22.4.2에서 서술한다.

7.7 기정파일허가권(umask)

파일과 등록부들을 만들 때 거기에 할당되는 기정허가권(default permission)들은 체계의 기정설정에 의존한다. 이 기정허가권들은 모든 사용자들에 의하여 만들어 진 파일들과 등록부들에 넘겨 진다.

- rw-rw-rw-(8진수 666)　　　　정규파일들에 대하여
- rwxrwxrwx(8진수 777)　　　　등록부에 대하여

그러나 이것들은 파일이나 등록부를 만들 때 보게 되는 허가권들이 아니다. 이 기정값은 사실 하나 이상의 허가권을 제거하기 위하여 **사용자마스크**(user mask)를 그 값에서 더는 방법으로 변환될수 있다. 이 마스크는 인수없이 umask를 사용함으로써 결정된다.

```
$ umask
022
```

이 수는 8진수인데 이 값은 그 파일의 기정값에서 덜면 666-022=644로 된다. 이것은 정규파일을 만들 때 보통 볼수 있는 기정허가권들(rw-r--r--)을 표현한다. 마찬가지로 기정등록부허가권도 rwxr-xr-x(777-022=755)로 된다.

umask설정은 그 설정이 적당한가를 확인하는 관리자에 의해서만 변경될수 있다. 두가지 실례를 아래에 보여 준다.

```
umask 666        모든 허가권들을 해제
umask 000        모든 읽기-쓰기허가권들을 설정
```

명심해야 할것은 누구도 심지어 관리자조차도 체계전반의 기정으로 지정되지 않은 허가권들을 설정할수 없다는것이다. 그러나 요구에 따라 chmod를 리용할수 있다. 파일과 등록부허가권들에 대한 umask설정의 효과를 표 7-4에서 보여 준다.

표 7-4.　기정허가권에 대한 umask설정의 효과

umask값	기정적인 파일허가권	기정적인 등록부허가권
000	rw-rw-rw-	rwxrwxrwx
666	---------	--x--x--x
777	---------	---------
022	rw-r--r--	rwxr-xr-x
046	rw--w----	rwx-wx--x
066	rw-------	rwx--x--x
222	r--r--r--	r-xr-xr-x
002	rw-rw-r--	rwxrwxr-x
026	rw-r-----	rwxr-x--x
062	rw----r--	rwx--xr-x
600	---rw-rw-	--xrwxrwx

7.8 파일소유권

목록의 세번째와 네번째 마당은 파일소유자와 그룹소유자를 보여 준다. 기정적으로 파일소유자는 그 파일을 만든 사람이다. 다음의 목록을 고찰하자.

```
-rwxrw-r--    1 julie   dialout        717    Sep 14 20:37 wall.html
```

julie만이 이 파일의 속성을 변화시킬수 있다. 하지만 julie가 dialout그룹의 한 성원인가? 그것은 그 파일을 만든 사용자에 의존한다. 만일 만든 사람이 julie라면 그의 그룹은 자동적으로 그룹소유자가 된다. 만일 그가 기정소유권형태를 변화시키지 않았다면 dialout는 julie그룹이여야 한다.

만일 사용자가 자기의 등록부에 이 파일을 복사한다면 그 복사판의 소유자로 되며 그의 그룹은 그룹소유권을 가지게 된다(그 그룹이 다른 경우에). 그 파일의 허가권은 그때 기정값 즉 파일을 만들 때 보게 되는 형식으로 변화된다. 소유자가 있음으로 하여 사용자는 복사판의 속성을 마음대로 조작할수 있다.

한편 dialout그룹의 어떤 성원이 이 파일에 쓰기를 진행할수 있다. 이것은 그룹성원들이 어떤 파일묶음을 읽거나 쓸수 있도록 하려고 하는 그룹대상과제에서 중요한 요구로 된다. 그러나 dialout의 성원(julie는 제외하고)들은 누구도 이 허가권들을 변화시킬수 없다.

참고

파일이 자기에게 속하는것인지 어떻게 아는가? 그 렬거에서 소유권렬을 보고 who am i지령으로 검사하시오. 만일 걸리는게 있다면 그 파일의 소유자라는것을 의미한다.

7.8.1 소유권세부정보는 어떻게 보관되는가(/etc/passwd와 /etc/group)

UNIX지령의 렬거와 출력에서 일반적으로 볼수 있는 사용자이름과 그룹이름은 순전히 사용자의 편리를 위해 제공된다. 체계는 수들만 리해한다. 사용자ID는 사실상 수(UID)이며 이 수는 /etc/passwd파일안에 보관된다. 그룹ID(GUID) 역시 /etc/passwd와 /etc/group안에 보관되는 수이다. 아래에 흔히 《통과암호》파일로 불리우는 /etc/passwd에 의한 내용이 제시되여 있다.

```
julie:x:508:100:julie andrews:/home/julie:/bin/csh
```

이것은 첫번째 마당에 사용자이름을 보여 주는 7개 마당으로 된 행이다. julie는 UID로서 508을, GUID로서는 100을 가지고 있다. 이 그룹ID의 이름은 /etc/group에서 찾을수 있다.

```
dialout:x:100:henry,image,enquiry
```

첫번째 렬은 그룹이름을 보여 주며 세번째 렬은 수로 된 그룹ID(GUID)를 가지고 있다. 사용자는 여러 그룹에 속할수 있는데 /etc/passwd에서 보여 주는 GUID가 **1차그룹**(primary group)이다. /etc/group는 **2차그룹**(secondary group)에 대한 사용자이름들을 보여 준다. 여기서 dialout는 henry와 다른 두 사용자들이 속한 2차그룹이다.

대부분의 지령들은 이 두 파일들을 리용하여 소유권세부정보를 현시하기전에 이 수들을 이름으로 번역한다. 그러나 이름대신에 수들을 표시하기 위하여 -n선택항목과 함께 ls지령을 사용할수 있다.

7.8.2 렬거안의 침입자

때때로 렬거의 소유권마당안에서 소유자와 그룹소유자의 이름대신에 수들을 보게 된다.

```
-rwxrw-r--    1    30    204        717 Sep 14 20:37 wall.html
```

이러한 문제들은 흔히 파일들이 다른 체계로부터 전송되였을 때 부닥치게 된다. 이 사용자는 아마 다른 체계에도 동일한 등록자리(slrifj UID와 GUID값은 다른)를 가지고 있을것이다. 여기서 수를 이름으로 번역하는것은 이 체계의 /etc/passwd와 /etc/group에 수들이 존재하지 않기때문에 진행되지 않았다. 체계관리자들은 여기서부터 교훈을 찾게 된다.

그림 7-1에서 보여 준 romeo의 홈등록부에 대한 렬거를 주시하면 또 다른 침입자(intruder)를 볼수 있다.

```
-rw-r--r--    3    juliet    metal    9156    Mar 12 1999    genie.sh
```

romeo의 등록부안에 juliet가 소유한 파일이 있다. 이것은 다음과 같은 원인에 의한것이 있다.

- 이 등록부가 객관쓰기가능한것이여서 juliet가 이 등록부에 파일을 만들었다.
- romeo가 cp -p 즉 파일의 속성들을 보존하는 지령으로 juliet의 등록부로부터 파일을 복사하였다.
- 파일이 다른 체계로부터 전송되였는데 그 체계에서 romeo는 이 기계상에서 juliet가 가지고 있는것과 꼭 같은 UID를 가지고 있다.

만일 romeo와 같은 어떤 사용자가 자기의 등록부안에서 그러한 파일들을 찾는다면 어떻게 되겠는가? romeo는 파일을 소유하지 않았으므로 그가 파일에 쓰기할수 없다는것을 주목하시오. 비록 wall.html이 204그룹의 성원들에 의하여 쓰기가능하더라도 romeo는 이 그룹의 성원이 아니다. romeo가 metal그룹에 속한다 해도 genie.sh는 그룹쓰기불가능하다.

romeo는 이 파일들을 지우거나 복사할수 있으며 허가권을 변경시킬수 있다. 또한 그는 체계관리자가 자기에게 소유권을 전송할것을 요구할수 있다. 이것은 다음에 론의되는 chown과 chgrp지령에 의해 수행된다.

참고

만일 두 기계에 어떤 사용자의 등록자리를 설치한다면 사용자ID(UID)는 물론 그룹ID(GUID)가 일치하는가를 확인하시오. 그래야 파일들이 두 체계들사이에 자유롭게 전송될수 있다.

7.9 파일소유권의 변경(chown, chgrp)

파일이나 등록부의 소유권을 조작하기 위한 수단으로서 두개의 지령 chown과 chgrp이 있다. 그것들은 오직 파일소유자에 의해서만 사용될수 있다. 두 지령의 문법은 다음과 같다.

chown <선택항목들> <새로운 사용자파일들>
chgrp <선택항목들> <새로운 그룹파일들>

chown에 대해서 보기전에 이 지령들과 함께 리용되여야 할 note파일을 렬거해 보자.

```
$ ls -l note
-rw-r--r--   1 romeo    metal      347    May 10 20:30 note
```

여기서 우리는 chown과 chgrp지령을 실행하는 사용자가 그 파일의 소유자인 romeo라고 가정하자. 이제 romeo가 juliet에게 이 파일의 소유권을 넘겨 주려 한다고 하자. 그러자면 파일들의 앞에 인수로서 새로운 사용자의 ID를 가지는 chown(change ownership)을 사용해야 한다.

```
$ chown juliet note; ls -l note
-rwxr----x   1 romeo    metal      347    May 10 20:30 note
```

이 지령에 의해서 파일소유권이 일단 juliet에게 넘겨 지면 원래의 소유권을 되찾을수 없다.

```
$ chown romeo note
```

```
chown: cannot change owner ID of note: Operation not permitted
```

romeo소유자에게 속한 파일허가권들은 지금 juliet에게만 적용될수 있다. 따라서 romeo는 그룹과 다른 사용자들에 대한 쓰기권한이 없으므로 파일 note를 더이상 편집할수 없으며 지울수도 없다(물론 그것을 복사할수는 있다).

chgrp(change group)지령은 파일의 그룹소유자를 변경시킨다. 다음의 실례에서 romeo는 dept.lst의 그룹소유권을 dba로 변경한다.

```
$ chgrp dba dept.lst ; ls -1 dept.lst
-rw-r-r--    1 romeo    dba       139    Jun 8 16:43 dept.lst
```

dba로 변경시킨후 다시 원래대로 회복하려고 한다면 어떻게 되겠는가?

```
$ chgrp metal dept.lst ; ls -1 dept.lst
-rw-r--r--    1 romeo    metal       139    Jun 8 16:43 dept.lst
```

그는 넘겨 준 그룹성원자격을 다시 얻는다. 그는 아직 소유자이므로 파일과 관련된 모든 권리들을 보유하기때문에 이것은 가능하다. chmcd와 같이 chown과 chgrp는 둘 다 재귀방식으로 조작을 수행하기 위하여 -R선택항목과 함께 작용한다. 이 세 지령들은 상급사용자(super user)들이 사용한다 해도 제한이 없다. 사실 상급사용자는 이 장에 론의된 모든 파일속성을 변경시킬수 있다.

참고

만일 어떤 대상과제의 성원들이 파일묶음을 읽거나 쓰기할수 있도록 하고 싶으면 그것들에 대한 공동그룹을 가지도록 체계관리자에게 요구하며 다음에 그룹허가권을 rwx로 설정한다. 이것을 수행하는(점착비트로) 보다 효과적인 방법을 22.4에서 서술한다.

Linux

chown과 chgrp의 사용제한

Linux에서 chown지령은 오직 상급사용자만이 리용할수 있다. romeo가 지령을 사용하려고 시도할 때 체계는 다음과 같이 거절한다.

```
$ chown henry note
chown: note: Operation not permitted
```

Linux사용자는 chgrp를 사용할수 있지만 그룹을 다른 그룹으로(자기가 속하는) 변화시키려 할 때에만 사용할수 있다. 만일 julie가 dialout와 uucp그룹에 속한다면 그는 chgrp를 리용해서만 이 두 그룹에 들어 갈수 있다. 그는 뿌리에 소유권을 줄수 없다.

7.10 파일변경시간과 접근시간

UNIX파일은 허가권과 소유권외에 그와 관련된 세가지 시간도장(time stamp)들을 가진다. 이 절에서는 그중 두가지(첫 두가지)를 론의한다.

- 파일의 마지막변경시간 ls -l로 보여 준다
- 파일의 마지막접근시간 ls -ln로 보여 준다
- 색인마디의 마지막변경시간 ls -lc로 보여 준다

파일의 내용을 변경할 때 마지막변경시간은 핵심부에 의해서 갱신된다. ls -l이 이 시간을 보여 주는데 -t선택항목은 변경시간순서로 파일을 보여 준다. 즉 마지막으로 변경된 파일은 첫번째로 놓인다.

```
$ ls -lt
total 278
-rw-r--r--    1 romeo    metal      10411    May 10 15:56 chap02
-rw-r--r--    1 romeo    metal      19514    May 10 13:45 chap01
drwxr-xr-x    1 romeo    metal         64    May  9 10:31 helpdir
-rw-rw-rw-    1 romeo    metal         84    Feb 12 12:30 dept.lst
-rw-rw-rw-    1 romeo    metal       9156    Mar 12  1998 genie.sh
```

파일은 또한 접근시간 즉 누군가가 파일을 읽거나 쓰기하거나 실행시킨 마지막시간을 가진다. 이 시간도 체계가 유지하며 파일의 내용이 변경될 때에만 설정되여 있는 변경시간과는 완전히 다르다.

ls의 -u선택항목은 파일의 접근시간을 나타낸다. ls -lu를 사용할 때 접근시간은 매 파일에 대해 나타나는데 정렬순서는 표준으로 된다(ASCII 순서맞추기렬). 하지만 -t선택항목이 -u와 함께 결합되면 파일들은 접근시간순서로 렬거된다.

```
$ ls -lut unit02 unit03 unit04
-rw-r--r--    1 romeo    metal      48527   Mar 20 23:17 unit04
-rw-r--r--    1 romeo    metal      39480   Feb 28 16:35 unit02
-rw-r--r--    1 romeo    metal      27183   Feb 13 14:57 unit03
```

만일 이제 cat로 unit03파일의 내용을 본다면 접근시간은 갱신되지만 변경시간은 갱신되지 않는다. 이것은 ls -lu지령을 다시 사용하면 명백해 질것이다.

```
$ ls -lu unit03                접근시간을 다시 검사한다
-rw-r--r--    1 romeo    metal       27183   Sep 30 23:30 unit03
$ date
Wed Sep 30 23:30:20 EST 1999
```

또한 -t와 -ut와 함께 -r선택항목을 사용할수 있는데 이런 경우에 파일들은 매 시간(변경 또는 접근)의 반대순서로 정렬된다.

파일의 변경과 접근시간에 대한 지식은 체계관리자에게 있어서 아주 중요하다. 체계관리자가 리용하는 대부분의 도구들은 어떤 특별한 파일이 여벌체계에 있는가 없는가를 결심하기 위하여 이 시간도장들을 참고하게 된다. 흔히 파일재보관편의프로그람(tar또는 cpio)으로 여벌체계에서 파일을 뽑아 낼 때(선택항목을 사용하여) 그 파일에는 시간이 틀리게 기입된다. 만일 이런 현상이 발생한다면 실지로 파일을 변경하거나 접근함이 없이 어떤 편리한 값으로 시간을 재설정하도록 touch를 사용할수 있다. touch는 다음절에 론의된다.

주해

파일의 변경시간을 놔두고 파일의 접근시간을 변화시킬수 있다. 반대로 파일을 변경할 때 일반적으로 그것의 접근시간도 물론 변경된다. 그러나 제외되는것이 있다. 즉 지령의 출력을 어떤 파일로 보낼 때(>와 >>기호로) 파일의 내용은 변경되지만 마지막접근시간은 변경되지 않은채로 남아있다. 이런 기능은 대부분의 UNIX와 Linux체계에서 보여 준다.

참고

cp로 파일을 복사할 때 무슨 현상이 일어 나는가? 기정적으로 복사판은 복사할 때에 설정되는 두가지 시간도장들을 가진다. 때때로 이와 같은 현상이 일어 나지 않을수도 있다. 이 경우에 시간도장들을 둘 다 보유하려면 cp -p(preserve)를 사용하시오. 이 선택항목은 또한 이미 있던 소유권형태를 보존한다.

7.11 시간도장의 변경(touch)

방금 론의된바와 같이 때때로 미리 정의된 값으로 변경시간과 접근시간을 설정하는것이 필요할수 있다. touch지령은 이 시간들을 변경시키는데 다음의 형식으로 사용된다.

```
touch <선택항목들> <식> <파일이름들>
```

touch가 선택항목 또는 식이 없이 리용될 때 이 시간들은 둘 다 현재시간으로 설정된다. 만일 그 파일이 존재하지 않으면 파일을 만들 때 덧쓰기되는 현상이 나타나지 않는다.

```
touch note                만일 이 파일이 존재하지 않으면 파일을 만든다
```

touch가 선택항목이 없이 식을 가지고 사용될 때 시간들을 둘 다 변경시킨다. 식은 8개의 수 즉 MMDDHHmm(월, 일, 시간, 분)형식으로 된 수로 구성된다. 요구에 따라 년문자렬을 2개 혹은 4개로 뒤붙이를 줄수 있다.

이제 emp.lst에 그 지령을 사용하자. 그러나 초기시간도장들을 본후에 사용하시오.

```
 $ ls -l emp.lst              변경시간
-rw-r--r--    1 romeo    metal       870    Mar 11 12:49 emp.lst
$ ls -lu emp.lst              접근시간
-rw-r--r--    1 romeo    metal       870    Jun 9 09:10 emp.lst
```

첫번째 지령은 변경시간을 보여 주지만 두번째것은 파일의 접근시간을 보여 준다. 여기서 touch는 시간을 둘 다 변경시킨다.

```
$ touch 03161430 emp.lst ; ls -l emp.lst
-rw-r--r--    1 romeo    metal       870    Mar 16 14:30 emp.lst
$ ls -lu emp.lst
-rw-r--r--    1 romeo    metal       870    Mar 16 14:30 emp.lst
```

또한 두 시간을 개별적으로 변경하는것이 가능하다. -m(변경)선택항목을 가지고 변경시간만 변화시킬수 있다.

```
$ touch -m 02281030 emp.lst ; ls -l emp.lst
-rw-r--r--    1 romeo    metal       870    Feb 28 10:30 emp.lst
```

-a(access)선택항목은 접근시간을 변경시킨다.

```
$ touch -a 01261650 emp.lst ; ls -lu emp.lst
-rw-r--r--    1 romeo    metal       870    Jan 26 16:50 emp.lst
```

체계관리자는 touch지령을 리용하여 파일을 증가여벌체계(incremental backup:변경된 파일만 여벌로 복사된다.)에 포함시키거나 거기에서 뽑을수 있도록 이러한 시간들을 수정한다. find지령이 일정한 시간이 지난 다음에 변경된 파일의 위치를 찾을 때 그 시간이 어떻게 설정되였는가에 따라 이런 파일들을 찾을수도 있고 못 찾을수도 있다.

7.12 파일체계와 색인마디

련결(link)에 대해서 론하기전에 UNIX체계에서 파일이 형성되는 방법에 대하여 먼저 보자. 파일의 속성을 보관하기 위하여 디스크의 어떤 령역이 항상 따로 설정된다. 우리가 론의한 모든 속성들은 지금까지 **색인마디**(inode)라고 부르는 표에 보관된다. 모든 파일은 한개의 색인마디를 가지며 **색인마디번호**(inode number)라고 부르는 번호에 의하여 접근된다. 파일의 색인마디번호는 단일한 파일체계에서는 유일하다.

이것은 우리에게 **파일체계**의 개념을 안겨 준다. 우리는 이 용어를 마치도 모든 파일들과 등록부들을 함께 가지고 있는 한개의 상부구조처럼 사용하였다. 드문히 그런 경우가 있는데 큰 체계에서는 결코 그렇지 않다. 하드디스크는 개별적인 파일체계로 분할되는데 매 파일체계는 자기의 뿌리등록부를 가진다. 만일 한개의 하드디스크가 3개의 파일체계를 가진다면 3개의 개별적인 뿌리등록부들을 가질것이다.

이 다중파일체계들중에서 하나는 기본적인것으로서 UNIX체계에서 가장 없어서는 안될 파일들을 포함한다. 이것이 바로 **뿌리파일체계**인데 적어도 어느 한 측면에서는 다른것들과 꼭 같다. 즉 그의 뿌리등록부는 또한 UNIX체계의 뿌리등록부이기도 하다. 기동시에 모든 보조파일체계들은 주파일체계에 첨부되여 사용자들이 하나의 파일체계로 느끼게 한다.

모든 파일체계는 색인마디를 모아 두기 위하여 표식된 개별적인 령역을 가지고 있다. 파일의 색인마디번호는 그 파일을 가지고 있는 파일체계에서만 유일하다.이것은 같은 색인마디번호를 가진 두개의 파일은 서로 다른 2개의 파일체계상에 있어야 한다는것을 의미한다.

이것들이 바로 련결에 대해서 리해하는데 필요한 지식이다. 색인마디와 다른 파일체계들에 대한 개념들은 제21장에서 상세히 취급한다.

7.13 련결(ln)

UNIX는 하나의 파일이 하나이상의 이름을 가지도록 허락하며 디스크에는 하나의 복사판을 보존한다. 이때 파일은 하나이상의 **련결**(link)을 가지고 있다고 말한다. 파일은 사용자가 주는 모든 이름들을 가질수 있지만 공통점은 그것들모두가 같은 색인마디번호를 가진다는것이다.

아래에서 보다싶이 목록의 두번째 렬로부터 파일의 련결수를 쉽게 알수 있다. 이 수는 보통 1이지만 이 파일은 2개의 련결을 가지고 있다.

```
-rw-r--r--    1 sumit    dialout     504    Oct 4 16:29 alias.sam
```

파일은 인수로서 두개의 파일이름을 가진 ln지령에 의하여 련결된다. 첫번째 파일이름은 실제로 존재하는 이름이며 두번째 파일이름은 우리가 제공한 별명이다. 다음의 지령은 현재 존재하고 있는 display.sh파일을 print.sh로 련결한다.

```
ln display.sh print.sh
```

ls의 -i선택항목은 파일의 색인마디번호를 현시한다. 여기서 첫번째 렬은 두 련결이 같은 색인마디번호를 가지고 있다는것을 보여 주는데 최종적으로 그것들은 사실상 하나이며 같은 파일이라는것을 증명하고 있다.

```
$ ls -li display.sh print.sh
```

```
29518 -rwxr-xr-x  2 romeo  metal   915 May  4 09:58 display.sh
29518 -rwxr-xr-x  2 romeo  metal   915 May  4 09:58 print.sh
```

련결수는 2개로 나타난다. show.sh라고 하는 또 다른 련결을 제공할수 있으며 그 수는 3개로 증가한다.

```
$ ln print.sh show.sh ; ls -li display.sh print.sh show.sh
29518 -rwxr-xr-x  3 romeo  metal   915 May  4 09:58 display.sh
29518 -rwxr-xr-x  3 romeo  metal   915 May  4 09:58 print.sh
29518 -rwxr-xr-x  3 romeo  metal   915 May  4 09:58 show.sh
```

비록 ls가 3개 파일들을 보여 준다고 해도 그것들은 사실상 한개이다. 파일은 2개의 별명을 가지고 있다. 련결은 모든 측면에서 다 같다. 즉 그것들은 같은 색인마디번호와 허가권, 시간도장들을 가지고 있다. 어느 한 련결로 만들어 진 변화들은 자동적으로 다른 곳에서도 가능하다.

련결은 자주 다른 등록부안에서 발생하는데 find지령을 사용하지 않는다면 그것들의 위치를 찾아 내기 어렵다. 실례로 다음의 지령은 다른 등록부에 있지만 같은 이름을 가진 련결을 제공한다.

```
ln toc.txt ../project5_safe/toc.txt
```

파일을 제거하려면 rm을 사용한다. 기술적으로 말해서 rm은 자동적으로 파일로부터 련결을 제거한다. 그러므로 다음의 지령은 련결 show.sh를 제거하며 련결수를 감소시킨다.

```
$ rm show.sh ; ls -l display.sh print.sh
-rwxr-xr-x     2 romeo     metal       915 May 4 09:58 display.sh
-rwxr-xr-x     2 romeo     metal       915 May 4 09:58 print.sh
```

rm을 한번 더 쓰면 련결수가 하나 감소된다. 련결수가 령으로 될 때 파일은 체계로부터 완전히 제거되는것으로 볼수 있다. 련결은 이렇게 돌발적인 지우기에 대처하여 어떤 보호를 제공한다. 만일 파일이 련결되여 있는데 무심결에 파일을 지우려고 rm을 사용하였다고 해도 적어도 아직 한개의 련결은 쓸수 있다. 파일은 아직 없어 지지 않았다.

련결을 어디에 사용하는가

어느때 파일을 련결해야 하는가? 다음과 같은 세가지 경우를 생각할수 있다.

1. 일반적으로 지정된 등록부(많은 프로그람들이 그 파일을 찾으려고 하는)안에 그 파일을 "가상적으로 배치"하기 위하여 련결을 사용한다. 말하자면 절대경로이름 /usr/local/bin/perl을 리용하여 perl을 호출하는 프로그람들을 가지고 있다. 체계를 갱신할 때 per1이 /usr/bin에로 옮겨 졌다는것을 알수 있다. 이 새로운 경로를 지적하기 위해 프로그람모두를 변경시키겠는가? 반드시 그렇지는 않다. 아래와 같이 /usr/local/bin안에 련결을 남긴다.

```
ln /usr/bin/perl /usr/local/bin/perl            필요한 뿌리허가권
```

2. C와 쉘프로그람작성언어는 프로그람이 자기 이름을 알고 있도록 하는 기능을 제공한다. 입력을 양식화하고 말단에 그것을 현시하는 display.sh프로그람을 생각하자. 지금 또 다른 프로그람 print.sh를 가지고 있는데 이 프로그람은 우와 류사한 일감을 수행하며 인쇄기에 인쇄한다고 하자. 두 프로그람은 실제로 공동부분을 가지므로 하나의 파일안에 모든 론리를 가지고 있는것으로 볼수 있다.
이 파일안에 있는 코드의 어떤 부분은 파일의 이름을 검사한다. 그다음 그것은 말단이나 인쇄기에 일감을 인쇄하기 위해 장치지정코드를 실행한다. 일단 프로그람이 개발되면 두가지 서로 다

른 이름을 가지도록 파일을 련결할수 있다. 프로그람을 보수하는것은 또한 두가지 일을 다 하는 파일이 하나만 있기때문에 쉽게 된다. 18.12에 이 특징을 사용하는 쉘스크립트가 있다

3. 련결은 또 다른데 쓸모 있다. 자주 사용하는 파일들이 다른 보조등록부에 배치되였다고 가정하자.

```
/home/henry/.profile
/home/henry/.elm/elmrc
/home/henry/Mail/received
```

모두 중요한 파일들이지만 길거나 기억하기 어려운 경로이름들을 가지고 있다. 모든 곳에서 이 경로이름들을 사용할 필요는 없다. 즉 이런 파일들의 련결들을 어떤 작업등록부 실례로 $HOME/work안에 배치한다.

```
cd $HOME/work
ln ../.profile prof.lnk
ln ../.elm/elmrc elmrc.lnk
ln ../Mail/received recd.lnk
```

상대경로이름을 사용하여 타자부하를 줄일수 있다. 이 파일들을 일단 련결하였다면 작업등록부안에서 그것들을 쓸수 있는것으로 생각할수 있다. .elm등록부안에 있는 elmrc파일을 편집하기 위하여 vi ../.elm/elmrc를 더이상 사용할 필요가 없다. 간단히 vi elmrc.lnk를 사용할수 있다. 이 파일들에 접근하는데는 경로이름이 필요 없다.

7.14 기호련결

앞에서 론의된것은 다음의 질문들을 제기한다. perl에 대하여 진행한것과 같은 련결을 만드는것은 파일을 너무 많이 련결하지 않는한 크게 문제될것이 없다. 그러나 /usr/local/bin이 초기에 100개의 프로그람을 포함하고 있었는데 그것들모두를 /usr/bin에 옮기면 어떻게 되겠는가? ln지령을 100번 리용하겠는가? 그리고 등록부 /usr/local/bin이 서로 다른 하드디스크상에 있다면 어떻게 되겠는가?

여기서 련결의 제한조건 즉 아래에 서술된 형태중에 한가지라도 부닥치게 된다. 련결은 2개의 심각한 약점을 가진다.

- 2개의 파일체계를 지나 파일을 련결할수 없다. 다시 말하여 만일 /usr파일체계안에 어떤 파일을 가지고 있다면 그것을 /home파일체계안에 있는 파일과 련결할수 없다.
- 같은 파일체계안에서조차도 등록부를 련결할수 없다.

기호련결(symbolic link)은 이러한 두가지 문제들을 다 극복한다. 다른 련결과 달리 기호련결은 실제로 내용을 가지고 있는 파일이나 등록부를 지적하는 등록부항목(entry)이다. 이 련결은 다른 련결보다 더 유연하므로 **연련결**(soft link)이라고도 한다. 이전의 절에서 론의한 련결은 **경련결**(hard link)이라고 한다.

/usr/local/bin안에 있는 100개의 프로그람을 련결하는 문제에 대답하려면 이 등록부들에 기호련결을 사용하여야 한다. 그 지령은 -s선택항목을 리용하는것을 제외하고 꼭 같다. /usr/local/bin이 /usr/bin을 지적하게 하자.

```
cd /usr/local
ln -s /usr/bin bin              두번째 파일은 존재하지 말아야 한다
```

기호련결은 경로이름을 포함하는 등록부항목이기때문에 디스크상의 공간을 차지하지 못한다. ln지령에 대하여 목적파일이름이 존재하지 않는가를 확인하시오. 이제 ls -l지령을 실행할 때 목록의 2개 렬은 서로 다르게 보여 준다.

```
$ pwd
/usr/local
$ ls -l
lrwxrwxrwx    1 root    root       8   Mar 1 23:53 bin -> /usr/bin
```

허가권마당의 l문자에 의하여 기호련결을 구별할수 있다. 표기법 bin -> /usr/bin은 bin이 경로이름을 등록부 /usr/bin으로 한다는것을 의미한다.

대부분의 체계들은 정규파일들을 련결하기 위해 경련결보다 오히려 기호련결을 사용한다. 다음의 ln지령은 같은 등록부에 있는 2개의 파일들을 련결하는데 ls의 -i(inode)선택항목은 또 다른 내용을 가진다.

```
$ ln -s note note.sym
$ ls -li note note.sym
9948 -rw-r--r--    1 henry     group        80 Feb 16 14:52 note
9952 lrwxrwxrwx    1 henry     group         4 Feb 16 15:07 note.sym -> note
```

두개의 파일들은 서로 다른 색인마디번호(따라서 한개의 련결수)와 파일크기를 가진다. 즉 그것들은 2개의 분리된 파일들이다. 그러나 note이지 note.sym은 아니며 이것은 사실 자료를 포함하고 있다. 여기서도 물론 상대경로이름을 사용할수 있다.

```
$ ln -s ../jscript/search.htm search.htm
$ ls -l search.htm
lrwxrwxrwx   1 sumit   dial   21   Mar 2 00:17 search.htm -> ../jscript/search.htm
```

왜 보통파일에 대해서 기호련결을 사용하였는가? 경련결과는 달리 기호련결은 파일체계들을 지나 파일들을 련결할수 있다. 이것은 두개의 하드디스크안에 있는 두개의 파일들이 기호련결로 련결될수 있다는것을 의미한다.

비록 기호련결이 등록부들을 련결할수 있다고 하더라도 보통파일과 같이 하나의 방법으로 동작하며 꼭 같은 방법으로 제거된다. 이것은 rm지령이 기호련결을 제거한다는것을 의미한다(그것이 등록부를 지적하더라도).

```
$ rm /usr/local/bin              보통파일
$ _
```

기호련결이 아니라 지적된 파일을 지우면 어떻게 되겠는가? 그러면 렬거에는 아무것도 지적되지 않으며 그것은 거기에 더이상 없다는것을 확인하려는 파일에 접근하려고 하는 경우에만 제기된다.

```
$ rm ../jscript/search.htm
$ ls -l search.htm
lrwxrwxrwx  1 sumit  dial  21 Mar 2 00:17 search.htm -> ../jscript/search.htm
```

```
$ cat search.htm
cat: search.htm: No such file or directory
```

기호련결은 UNIX체계에서 광범히 사용된다. 체계파일은 끊임없이 판본의 증가와 함께 위치들을 변화시킨다. 그러므로 모든 프로그람들이 여전히 원래 있었던 파일들에 도달하는가를 확인하여야 한다.

기호련결과 함께 cd를 리용할 때 내장된 pwd지령은 등록부를 얻는데 사용된 경로를 보여준다. 이것은 실지등록부와 반드시 같지는 않다. 《실지》위치를 알려면 외부지령 /bin/pwd를 사용하여야 한다.

7.15 파일들의 위치찾기(find)

find는 체계의 쓸모 있는 도구들중의 하나이다. 이 지령은 이름에 의해서 또는 하나이상의 파일속성을 비교하면서 파일을 찾기 위해 등록부나무를 재귀적으로 조사한다. 또한 그것은 복잡한 지령행을 가지고 있는데 만일 UNIX가 왜 많은 사람들로부터 비난을 받는가에 대해 생각해 본적이 있다면 수수께끼같은 find문서를 보았을것이다. 그러나 find의 인수를 다음과 같은 3개의 구성요소로 해체해 보면 쉽게 다룰수 있다.

```
find path_list selection_criteria action
```

그림 7-4는 대표적인 find지령의 구조를 보여 준다. 그 지령은 다음과 같은 방법으로 등록부나무를 정확하게 조사한다.

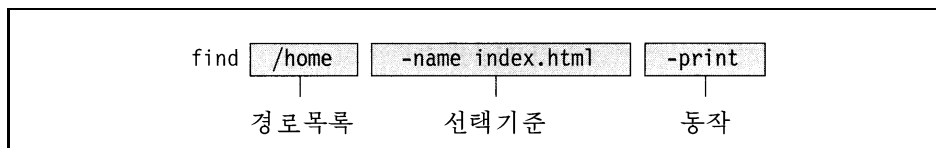



그림 7-4. find지령의 구조

- 첫째로 경로목록안에 지적된 등록부에서 모든 파일들을 재귀적으로 조사한다. 여기서는 /home으로부터 탐색이 시작된다.
- 그다음 한개 또는 그이상의 선택기준에 비추어 매개 파일을 비교한다. 이것은 항상 - 연산자 인수(-name index.html)형식으로 구성된다. 여기서 find는 그 등록부가 index.html이름을 가진 파일을 가지고 있다면 그 파일을 선택한다.
- 마지막으로 선택된 파일들에 대해 어떤 동작을 수행한다. 동작 -print는 말단에 find출력을 간단히 표시한다.

모든 find연산자들은 -로 시작되지만 경로목록은 결코 하나만 포함할수 없다. 사용자는 경로목록과 한개 또는 그이상의 파일들을 비교하기 위한 다중선택조건들로 동작하는 보조등록부들을 제공할수 있다. 이것은 처음에 리용하던것과는 복잡하지만 실지로 어떤 조건하에서 파일을 선택하게 해주므로 모든 사용자들이 정통해야 할 프로그람이다.

core로 이름 지어 진 모든 파일(디스크상에 옮겨 진 처리영상)을 찾기 위해서 먼저 find지령을 실행하자.

```
$ find / -name core -print
/home/romeo/scripts/core
```

```
/home/andrew/scripts/reports/core
/hoem/juliet/core
```

탐색은 뿌리등록부로부터 시작하므로 find는 파일의 절대경로이름을 현시한다. 또한 파일이름의 어떤 묶음을 얻어 내기 위하여 이름안에 메타문자(*와 같은것)도 사용할수 있는데 그 경우에는 인용부호로 막아 주어야 한다.

```
find . -name "*.lst" -print                인용부호를 잊어서는 안된다
```

이 지령은 현재등록부나무로부터 확장자 .lst를 가지고 있는 모든 파일이름들을 얻어 낸다(.은 현재등록부이다). 일단 *와 같은 특수기호들을 사용하는 방법(8.2)을 알고 있으면 실지 힘 있는 도구로서 find를 사용할수 있다. 이런 방법으로 대문자로 시작되는 모든 파일이름을 탐색할수 있다.

```
find . -name '[A-Z]*' -print               외인용부호도 쓸수 있다
```

이러한 점에서 find는 Windows의 계수지령보다 더 위력하다.

주해

find에서는 -name인수와 함께 메타문자들의 모임을 쓸수 있다. 즉 *가 그중의 하나이다. 쉘은 또한 꼭 같은 설정(8.2)을 사용하는데 그것들을 사용하는것을 배운 다음에는 아주 쉽고 쓸모있는 find로 패턴을 찾을수 있다. 실례로 우의 경우에 관계없이 모든 .doc파일의 위치를 알아 내기 위하여 find를 사용할수 있다.

```
find -name "*.[Dd][Oo][Cc]" -print
```

find는 쉘에서 사용된것과 마찬가지의 패턴정합기능을 제공하는 유일한 UNIX지령이다.

find선택항목

-name은 선택기준을 조작하는데 리용되는 유일한 연산자는 아니다. 즉 다른것들도 많이 있다. 표 7-5는 파일의 속성을 나타내는 ls -lids와 꼭 같은 렬로 find의 선택기준을 보여 준다. 대부분의 선택항목들은 아주 직관적이다. find를 리용하기 쉽게 하는 일부 방법들을 보자.

파일형과 허가권(-type 와 -perm)

-type연산자뒤에 f, d, l문자들을 놓으면 보통파일과 등록부 그리고 기호련결형태의 파일들을 선택한다. 아래에 홈등록부나무의 모든 등록부들을 알아 내는 방법을 준다.

```
$ find /home/henry -type d -print
/home/henry
/home/henry/.elm
/home/henry/Mail
/home/henry/.netscape
```

또한 특수한 허가권모임을 가지고 있는 파일들을 알아 내기 위해 -perm연산자를 사용할수 있다. 실례로 -perm 666은 모든 부류의 사용자들에 대한 읽기와 쓰기허가권을 가지고 있는 파일들을 선택한다. 이러한 파일들은 보안이 안된 파일들이다. 때때로 등록부만을 탐색하는것으로 제한하기 위해 2개의 연산자들을 사용하고 싶을것이다.

```
find /home/herry -perm 777 -type d -print
```

여기서 우리는 모든 사람들에게 모든 접근권을 제공하는 등록부 즉 보통파일에 의해 리용되는

-perm 666보다 보안이 더 안된 등록부를 찾고 있다. 그것은 우에서 실행되는 AND론리곱하기조건이다. 즉 find는 두 선택기준(-perm과 -type)에 꼭 들어 맞는 파일만을 선택한다.

표 7-5. find에 리용되는 연산자(+는 반대의미인 -로 교체될수 있다.)

선택기준	의 미
-inum n	색인마디번호를 가진 파일을 선택한다
-type x	x형의 파일을 선택한다. 즉 x는 f(보통파일), d(등록부), l(기호련결)일수 있다
-perm nnn	8진수허가권들이 nnn에 꼭 맞는 파일을 선택한다
-perm -400	다른 비트들이 있든 상관없이 소유자가 읽기허가권을 가진 파일을 선택한다
-links n	n개의 련결을 가진 파일을 선택한다
-user usname	usname이 소유한 파일을 선택한다
-group gname	그룹 gname이 소유한 파일을 선택한다
-size +x[c]	크기가 x개의 블로크들(c로 지정되는 경우 x개의 문자들)이상인 파일을 선택한다
-mtime -x	x날자전에 변경된 파일을 선택한다
-mmin -x	x분전에 변경된 파일을 선택한다(Linux에서만)
-newer flname	flname후에 변경된 파일을 선택한다
-atime +x	x날자이후에 접근된 파일을 선택한다
-amin +x	x분이후에 접근된 파일을 선택한다(Linux에서만)
-name flname	flname파일을 선택한다
-iname flname	우와 같지만 대문자를 구별한다(Linux에서만)
-follow	기호련결이 따르는 파일을 선택한다
-prune	정합되였다면 등록부로 내려 가지 못한다
-mount	다른 파일체계를 보지 못한다
동 작	**의 미**
-exec cmd	{} \이 뒤따르는 UNIX지령 cmd를 실행
-ok cmd	cmd가 사용자구성다음에 실행된다는것을 제외하고 -exec와 같다
-print	말단우에 선택된 파일을 인쇄
-ls	선택된 파일에 ls -lids지령을 실행

기호련결(-follow)

만일 파일이 등록부를 지적하는 기호련결이라면 find는 기정적으로 그것이 경로목록에 없는 경우 그 등록부를 들여다 보지 않는다. 이 점을 증명하기 위하여 먼저 night.gif파일을 포함하고 있는 ../jscript 등록부를 기호적으로 련결하자.

```
ln -s ../jscript jscript
```

우리는 이제 어미등록부에서 꼭 같은 이름의 등록부를 지적하는 기호련결 jscript를 가진다. 우리는 night.gif를 찾기 위하여 find를 실행할 때 그 등록부를 들여다 보지 않는다.

```
$ find . -name night.gif -print
$ _
```

그렇지만 -follow연산자와 이것을 결합하면 find는 다음과 같이 파일의 위치를 찾아 낸다.

```
$ find . -name night.gif -follow -print
```

```
./jscript/night.gif
```

상대경로이름 find가 현시되는데 그것은 경로목록자체가 상대적(.)이기때문이다.

변경시간의 정합(-mtime)

파일의 변경시간을 정합하기 위하여 -mtime연산자를 사용할수 있다(-atime은 접근시간을 정합한다). 이 지령은 2일전에 변경된 파일들 즉 현재등록부로부터 시작된 파일들을 보여 준다.

```
$ find . -mtime -2 -print
.               현재등록부도 포함한다
./unit13
./unit15
./unit14
```

우에서 볼수 있는바와 같이 find는 반드시 ASCII순서로 정렬된 목록을 현시하는것은 아니다. 파일들이 현시되는 순서는 파일체계의 내부적인 구조에 의존한다. 더 나아가서 꼭 같은 선택항목을 두번 사용함으로써 범위를 표현하는 탐색조건을 좁힐수 있다.

```
$ find . -mtime +2 -mtime -5 -ls
37353     2 -rwxr-xr-x   1 sumit   dialout       26 Apr 21 21:38 ./toc.sh
37392   296 -rwxr-xr-x   1 sumit   dialout   150426 Apr 20 23:39 ./session4
37393   146 -rw-r--r--   1 sumit   dialout    73728 Apr 20 23:10 ./session5
37394    38 -rw-r--r--   1 sumit   dialout    17946 Apr 20 23:10 ./session6
```

여기서 find는 2일이후와 5일이전에 변경된 파일들의 목록을 표시하기 위해 -ls동작을 리용한다. -ls 즉 동작구성요소는 ls -lids지령을 실행하며 색인마디번호(첫번째 렬)와 512Kbyte블로크(두번째 렬)단위로 파일크기를 보여 준다.

-print와 -ls 외에 find는 간단히 론의되는 2개의 다른 동작구성요소들을 리용한다.

주해 +365는 365일보다 더 크다는것을 의미하며 -365는 365일보다 작다는것을 의미한다. 365를 정확히 지적하려면 365를 사용하시오

find는 또한 론리곱하기와 론리더하기조건을 렬거하기 위하여 각각 -a와 -o연산자를 사용한다. 2개의 선택기준의 사용은 일반적으로 론리곱하기조건을 표현한다. 그러나 perl과 스크립트의 쉘판본의 위치를 둘 다 알아 내려면 -o를 사용하는것이 필요하다.

```
find /home -name binary.pl -o -name binary.sh -print
```

주해 -a 또는 -o연산자를 사용하지 않고 선택기준이 련속 렬거될 때 그것들은 론리곱하기조건으로 번역된다. 이전의 실례에서 -a를 사용하지 않고 -mtime을 두번 사용하였다. 즉 론리곱하기를 암시한다. 확인하려면 다음의 두 지령을 사용하고 어떤것이 차이나는가를 알아 보시오.

```
find . -mtime +2 -mtime -5 -1s
find . -mtime +2 -a -mtime -5 -1s
```

선택된 파일에 동작을 가하기(-exec와 -ok)

이 연산자들은 말단에 파일의 목록(>연산자를 사용하여 파일안에 보관할수 있는 목록)을 만든다. 어느 실례에서 우리는 파일의 목록을 현시하였다(-ls로). 그러나 실생활에서 그것들을 지우는것과 같은 일부 동작을 가하고 싶을것이다. 이것은 -exec연산자를 가지고 수행되는데 {} \순서로 실행되고 완료되는 지령뒤에 놓인다. 다음의 지령은 Nar/preserve안에 있는 한달이상된 모든 림시파일들을 제거한다.

```
find /var/preserve -mitme +30 -exec rm -f {} \;            {}는 파일이름을 표시한다
```

-exec는 어떤 UNIX지령의 실행을 허가한다. 여기서 파일이름은 {}로 렬거되며 rm은 제거를 요구한다. exec는 \으로 완료된다. 이 연산자의 사용법은 아주 수수께끼 같지만 잊어 먹을 걱정이 없다.

재확인이 없이 파일을 제거하는것은 매우 위험하므로 대화적인 삭제를 위해 -exec대신에 -ok연산자를 사용할수 있다. 이것은 find가 선택적으로 2000개의 블로크보다 더 크며 180일동안 한번도 접근되지 않은 파일들을 제거하는 방식이다.

```
$ find /home -size +2000 -atime +180 -ok rm {} \;
< rm ... /home/romeo/README >? y
< rm ... /home/juliet/README >? n
......
```

매 파일은 결심을 요구한다. y는 지령을 실행하며 다른 응답은 파일을 지우지 않고 그대로 남긴다.

find는 체계관리자의 도구인데 제22장에서 어떤 과제들을 수행하기 위해서 리용되는 find에 대해서 취급한다. 특히 이 지령은 파일들을 여벌복사하고 xrags지령과 합동하여 사용하는데 적합하다(22.10).

주해

find와 함께 -exec 또는 -ok를 사용할 때 {}를 가지고 파일이름을 나타내지만 \으로 UNIX 지령행을 끝낸다는것을 잊지 마시오.

Linux

UNIX의 find는 오직 -print연산자가 리용되여야만 파일이름을 현시한다. 그러나 GNU의 find는 이 선택항목이 필요하지 않다. 즉 그것은 기정적으로 인쇄된다. 또한 경로목록이 필요 없다. 즉 기정적으로 현재등록부만 사용한다. 다시 말해서 지령행 find -name "*.java"는 현재보조등록부나무에서 모든 .java 파일들을 검사하고(match) 인쇄한다. 문제를 더 《간단히》하려면 GNU find에 그 어떤 인수도 주지 않는다. 즉 자체로 리용된 find는 현재등록부나무에서 모든 파일들을 재귀적으로 현시한다.

-iname연산자는 대소문자를 구별한다. -mmin과 -amin선택항목은 비교하기 위한 단위로서 -mtime과 -atime이 리용하는 시간이 아니라 분을 리용한다.

요 약

우리는 어떤 형식으로 파일들을 보기 위하여 ls지령을 사용할수 있다. 즉 여러개의 렬로(-x), 등록부와 파일들을 구별하기 위해서(-F) 그리고 점으로 시작하는 숨겨 진 파일들을 현시하기 위해서(-a) 사용할수 있다. 정렬순서를 반대로(r) 할수 있으며 재귀적인 목록(-R)을 얻을수 있다.

기정적으로 ls는 ASCII순서맞추기렬로 파일들을 정렬시키는데 수자가 먼저 놓이고 그다음 대문자, 소문자순서로 정렬된다.

UNIX파일은 일부 속성들을 가지고 있는데 그것들중 7개가 ls -l에 의하여 현시된다. 그것들은 허가권, 련결, 소유자와 그룹소유자, 크기, 날자 그리고 마지막변경시간, 파일이름이다. ls -ld는 등록부속성들을 보여 준다.

바이트로 된 파일의 크기는 파일이 디스크를 차지하고 있는 실제공간이 아니라 그것을 포함하고 있는 바이트수이다. 1byte를 가진 파일은 실제적으로 디스크에 1024byte를 차지할것이다.

파일은 읽기, 쓰기 또는 실행가능한 허가권들을 가질수 있는데 파일의 소유자, 그룹소유자는 물론 다른 사용자들에 대한 3가지 허가권묶음이 있을수 있다.

chmod는 이 허가권들을 변경시키는데 사용되며 오직 파일소유자에 의해서만 사용된다. 허가권들은 + 또는 -기호와 함께 사용될 때는 상대적일수 있고 8진수들로 사용되면 절대적일수 있다.

등록부허가권의 의미는 보통파일들과 차이난다. 읽기허가권은 등록부안에 보관된 파일이름이 읽기가능하다는것을 의미한다. ls는 등록부파일을 읽는것에 의하여 동작한다. 쓰기허가권은 등록부안에서 파일을 만들거나 제거하도록 허락한다. 실행허가권은 등록부로 "cd"하게 해준다.

umask설정은 파일이나 등록부를 만들 때 사용하는 기정허가권들을 결정한다. 파일은 소유자 대체로 파일을 만든 사용자의 이름을 가진다. 파일은 또한 그룹에 의해 소유되기도 하는데 기정적으로는 그 사용자가 속하는 그룹이다. 소유자만이 파일속성들을 변경시킬수 있다. chown과 chgrp지령은 소유권과 그룹소유권을 변환하는데 사용된다.

파일은 변경시간과 접근시간을 가지고 있다. ls는 파일의 변경시간(-t) 혹은 접근시간(-ut)에 따라 파일을 정렬할수 있다. touch는 어떤 임의의 값으로 이 시간을 변경한다.

다중파일체계에서 매 파일체계는 자기의 뿌리등록부를 가지며 하나의 파일체계형식으로 체계기동을 병합한다. 파일은 색인마디번호에 의해 식별되는데 그 속성들은 색인마디에 보관되여 있다. 색인마디번호는 ls -i에 의해 현시된다.

파일은 하나이상의 이름을 가질수 있다(경련결). ln으로 파일들을 련결할수 있으며 rm으로 련결을 제거한다. 련결은 꼭 같은 색인마디번호를 가진다. 련결된 파일은 그것이 호출되는 이름에 따라 두개의 개별적인 지령처럼 동작할수 있다.

기호련결(연련결)은 다른 파일체계에 또 다른 파일이 있다고 할지라도 그 파일의 위치를 지적하는 등록부항목이다. 경련결과는 달리 연련결은 등록부들도 련결할수 있다. 기호련결은 rm으로 제거되고 ln -s로 생성된다.

find는 임의의 파일속성으로 될수 있는 어떤 기준에 만족하는 파일들을 찾는다. 파일은 형(-type), 이름(-name), 크기(-size), 허가권(-perm) 또는 시간도장(-mtime 과 -atime)들로 지적될수 있다. 어떤 UNIX지령은 확인을 가지고 혹은 확인없이 선택된 파일들(-exec와 - ok)상에서 실행될수 있다.

시험문제

1. ASCII순서맞추기렬로 정해 진 정렬순서는 무엇인가?

2. 어느 ls선택항목이 등록부와 실행파일들을 따로따로 표식하는가?
3. 숨겨 진 파일이란 어떤것인가?
4. 파일의 렬거를 무엇이라고 볼수 있는가?
5. 전체 체계의 모든 파일들과 등록부들을 어떻게 전부 렬거하겠는가?
6. 어미등록부의 파일들을 어떻게 보여 주는가?
7. 목록으로부터 등록부들을 어떻게 식별하는가?
8. 누가 파일 또는 등록부의 속성을 변화시킬수 있는가?
9. ls -l은 모든 파일들을 보여 주는가?
10. 어느 때 목록에서 변경시간(날자는 아니다.)을 보여 주지 않는가?
11. 그룹성원들이 파일을 제거할수 있게 하려면 무엇이 필요한가?
12. 파일을 만든 다음 기정파일허가권을 rw-r--r--로 가정하고 소유자에 대해서 모든 허가권들을 할당하며 다른 사용자들의 허가권들을 제거하려고 하는데 어떻게 할것인가?
13. 그룹과 다른사용자들에 대한 파일의 쓰기허가권을 제거하였는데 아직도 파일을 지울수 없다. 그 현상은 왜 그런가?
14. 파일의 색인마디번호를 어떻게 현시하는가?
15. 소유권과 그룹소유권의 세부정보들은 어디에 보관되는가?
16. 현재등록부안에 있는 모든 파일들의 소유권을 재귀적으로 다른 사람에게 이전시키시오.
17. 파일의 변경시간을 9월 30일 오전 10시 30분으로 수정하시오.
18. touch foo지령은 무엇을 하는가?
19. 색인마디는 무엇을 보관하는가?
20. 파일이 3개의 련결을 가진다는것은 무엇을 의미하는가?
21. 련결된 파일을 어떻게 제거하는가?
22. 체계안에 있는 모든 .html과 .java파일들을 알아 내기 위한 지령을 작성하시오.

련습문제

1. ls bar를 실행시켰을 때 10개의 파일들의 목록(bar가 그중의 하나이다.)이 나타났다. 어떻게 이렇게 될수 있는가?
2. 실행파일들과 등록부들에 특수한 표식을 하면서 여러개의 렬로 모든 파일들을(숨은 파일들도 포함해서) 재귀적으로 이떻게 표시하는가?
3. 소유자는 파일의 그룹소유자로서 같은 그룹에 속하는가?
4. 파일은 1026byte를 가진다. 파일은 몇바이트의 디스크공간을 차지하는가?
5. 현재등록부의 속성을 어떻게 보여 주는가?
6. 등록부크기는 왜 보통 작은가?

7. 만일 파일이 소유자에 대한 쓰기허가권을 가지지 못하였다면 그 파일을 제거할수 있는가?

8. 다음의 허가권들을 8진수표시로 보여 주시오.

   (1) rwxr-xrw-   (2) rw-r-----

9. 8진수값 567, 623에 대한 허가권문자렬은 무엇인가?

10. 다른 사용자의 등록부로부터 foo파일을 복사하려고 하는데 cannot create file foo라는 오유통보문을 받았다. 그는 자기 등록부의 쓰기허가권을 가지고 있다. 무엇이 원인으로 되며 어떻게 그 파일을 복사하는가?

11. chmod -w foo는 무엇을 하는가?

12. 만일 등록부가 허가권 777을 가지고 있으며 그안에 있는 파일이 허가권 000을 가지고 있다면 보안과 관련하여 중요한것은 무엇인가?

13. 자기가 가지고 있는 파일들의 이름을 누구도 볼수 없다는것을 확인하려면 무엇을 하여야 하는가?

14. 만일 등록부를 변경할수 없다면 그 원인으로 될수 있는것은 무엇인가?

15. 다른 사용자등록자리로 파일을 복사할 때 어느 파일의 속성이 변화되는가?

16. 그 속성들을 보존하면서 파일을 복사하자면 어떻게 해야 하는가?

17. 만일 소유자가 어떤 파일에 대한 쓰기허가권을 가지지 못하였지만 그가 속한 그룹은 가지고 있다면 그는 그것을 편집할수 있는가?

18. 소유권을 포기하려고 할 때 chown은 chgrp과 어떻게 다른가?

19. 만일 파일을 변경시키고 그것을 취소하고 보관한후에 편집기에서 탈퇴하였다면 그 파일이 변경되였다고 볼수 있는가?

20. 프로그람이 오늘 실행되였는지 어떻게 알아 낼수 있는가?

21. ls -l과 ls -lt사이의 차이점은 무엇인가?

22. ls -lu와 ls -lut사이의 차이점은 무엇인가?

23. 두개의 파일이 복사판인지 련결인지 어떻게 증명할수 있는가?

24. 경련결의 두가지 기본결함들은 무엇인가?

25. $HOME/progs안에 다른 프로그람들에 의하여 호출되는 몇개의 프로그람들을 가지고 있다. 이제 이 프로그람들을 $HOME/internet/progs에로 옮기기로 결심하였다. 사용자들이 이 변화를 모른다는것을 어떻게 담보하는가?

26. 체계관리자는 사용자들이 만든 모든 파일들이 기정허가권 rw-rw----를 가질것이라는것을 어떻게 담보하는가?

27. /bin과 /usr/bin등록부에서 z로 시작되는 모든 파일이름들을 찾아 내시오.

28. /oracle등록부나무에서 logrn.sql파일의 위치를 찾아 내는데 오직 find만 사용하고 그다음 자기의 등록부에 그것을 복사하시오.

29. 24시간전에 변경된 모든 파일들을 어미등록부밑에 있는 posix등록부에로 이동하는데 find를 사용하시오.

제 8 장. 쉘

이 장에서는 사용자와 UNIX체계사이에 위치하고 있는 매개물을 소개한다. 그것을 쉘이라고 부른다. 사용자가 UNIX에서 진행하는 모든 작업은 이 매개물이 처리한다. 쉘은 사용자의 몸짓으로부터 모든것을 직접 리해하는 훌륭한 서기와도 같이 우리가 알 필요가 없는 특별한 수단으로 그것들을 수행한다. 쉘은 지령처리기이다. 즉 사용자가 기계에 준 명령을 처리한다.

이 장은 이 책에서 가장 중요한 장의 하나인데 UNIX설계자의 일부 기본사상을 반영하였다. 여기서 강조하는 개념들은 완전히 리해하여야 한다. 이 개념들은 Bourne쉘에 기초하고 있는데 그 이름은 개발자 스티브 본(Steve Bourne)의 이름을 딴것이다. 이 쉘은 UNIX체계와 함께 발전해 온 가장 오래된 쉘이다. 사용자들은 아마 이 쉘을 리용하지 않겠지만 Bourne쉘은 모든 쉘들의 가장 작은 《공통분모》이며 그의 대부분의 기능들은 현대쉘들에서도 쓰인다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 쉘이 지령에 대해서 어떤 처리를 하는가를 리해한다(8.1).
- 통용기호를 리용하여 파일이름들을 정합한다(8.2).
- \을 리용하여 특수문자의 의미를 해제한다(8.3)
- 외인용부호와 겹인용부호를 리용하여 문자묶음을 보호하며 그것들사이의 차이점을 리해한다(8.4).
- echo지령에 확장문자렬을 사용한다(8.5).
- 흐름에 대하여 그리고 쉘이 그것들을 파일로 취급하는 방법에 대하여 배운다(8.6).
- 표준출력의 방향을 파일에로 돌린다(8.6.1).
- 표준입력이 파일로부터 나오도록 지적한다(8.6.2).
- 표준오유의 방향을 파일에로 돌린다(8.6.3).
- /dev/null과 /dev/tty파일들의 의미를 리해한다(8.7).
- 려파기의 속성과 |가 두개이상의 지령들을 련결하기 위한 관흐름을 설정하는데 리용되는 방법을 배운다(8.8).
- 지령대입을 리용하여 지령들을 다른 지령들의 지령행에 삽입한다(8.10).
- 쉘변수들의 속성을 배운다(8.11).
- 쉘스크립트에서 지령들이 어떻게 묶어 지는가를 배운다(8.12).

주해

사용자들은 이러한 요구를 몰라도 아마 지금 광범히 리용되는 쉘들인 C쉘, Korn쉘, bash들중의 하나를 리용하고 있었을것이다. Korn과 bash는 Bourne의 상위모임이므로 Bourne에 적용되는것들은 역시 이 쉘들에도 적용된다. 그러나 이 장에서 론의하는 쉘의 일부 기능들은 C쉘에 적용되지 못한다. 그에 대해서도 서술한다.

자기가 리용하는 쉘에 대하여 알려면 echo $SHELL지령을 호출하시오. 그 출력은 /bin/sh(Bourne쉘), /bin/csh(C쉘), /bin/ksh(Korn쉘), /bin/bash(bash쉘)을 보여 줄것이다. 이 지령은 현 단계에서 리용하고 있는 쉘에 대하여 알려고 하는데 도움을 준다

8.1 지령처리기로서의 쉘

UNIX에 가입할 때 프롬프트가 보인다. 프롬프트는 $, % 등일것이다. 이것은 사실 이 장의 기본문제는 아니다. 비록 그것이 거기서 아무 일도 진행됨이 없이 나타날수 있다 하더라도 어떤 UNIX지령이 사실상 진행되고 있다. 그 지령이 **쉘**이다. 그것은 체계에 가입하는 순간에 동작을 시작하며 탈퇴할 때 완료된다.

사용자가 지령을 줄 때 쉘은 그 정보를 얻기 위한 첫번째 매개물이다. 쉘은 사용자들의 요구를 받고 해석한다. 그 요구란 바로 우리가 건으로 입력한 UNIX지령들이다. 쉘은 지령행을 검사하고 재편성한 다음 실행작업을 핵심부에 맡긴다. 핵심부는 이 지령들과 체계안에 있는 모든 프로쎄스들을 위하여 하드웨어를 조종한다. 이것은 UNIX설계와 원리의 우점중의 하나이다.

쉘은 일반적으로 잠 자고 있는데 프롬프트에 건입력이 진행될 때 깨여 난다(**잠자기**(sleeping), **기다리기**(waiting), **깨여나기**(waking)는 UNIX계에서 받아 들인 용어이다). 이 입력은 사실 쉘을 표현하는 프로그람(Bourne쉘은 sh)에로의 입력이다. 대체로 다음의 동작들이 쉘에 의해 수행된다(그림 8-1).

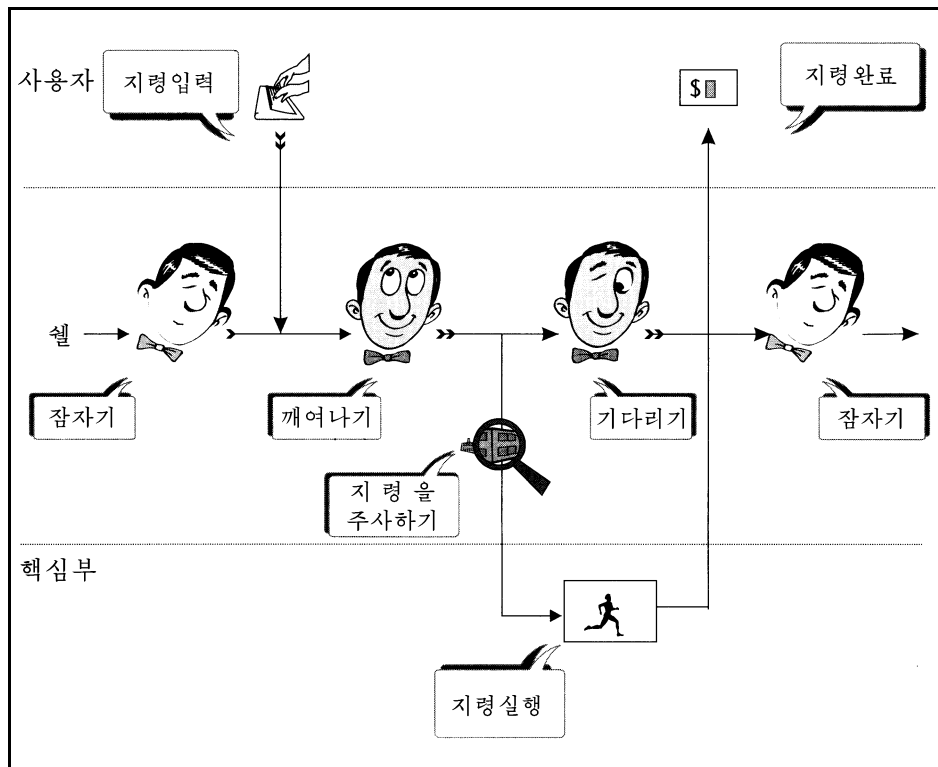



그림 8-1. 쉘의 해석주기

- 쉘은 프롬프트($ 혹은 다른것)를 발생시키고 지령이 입력될 때까지 잠을 잔다.
- 지령이 입력된후 쉘은 일부 특별한 의미를 가진 특수문자(이 장에서 론의하는 메타문자)들에 대하여 지령행을 주사한다. 생략된 지령행들이 허락되므로(rm *와 같이 모든 파일들을 지적하기 위하여 *를 사용하는것처럼) 쉘은 지령이 작용하기전에 생략을 확장시켰는가 확인하여야 한다.
- 다음 간단한 지령행을 만들고 실행을 위하여 그것을 핵심부에 보낸다. 쉘은 지령이 실행되고 있는 동안 아무런 작업도 할수 없으며 그것이 완성될 때까지 기다려야 한다.
- 일감이 완성된 다음에 프롬프트가 다시 나타나며 쉘은 다음《주기》를 시작하는 잠 자는 상태로

돌아 간다.

쉘은 보통 메타문자들이 지령에 대해서 아무것도 서술하지 못하기때문에 그 메타문자들을 해석하여야 한다. 이 장에서는 기본적으로 쉘의 해석적인 역할에 대해서 본다.

8.2 패턴정합과 통용기호

앞의 장들에서는 인수로서 하나이상의 파일이름(실레로 cp chap01 chap02 chap03 progs)들을 가진 지령들을 사용하였다. 흔히 지령행에 일부 류사한 파일이름들을 입력하는것이 필요할것이다.

```
ls -l chap chap01 chap02 chap03 chap04 chapx chapy chapz
```

여기서 파일이름들이 공통문자렬 chap를 가지고 있으므로 이 문자렬을 반복적으로 리용하는 길다란 지령행은 오히려 랑비적인것으로 보인다. 왜 우리는 한두개의 문자들로 문자렬 chap를 반영하는 하나의 패턴들을 가질수 없는가? 쉘은 이러한 해결책을 제공한다.

쉘은 일부 문자들을 특수하게 인식한다. 어떤 류사한 파일이름들의 모임을 정합할수 있는 개괄적인 패턴이나 모형을 고안하기 위하여 그것들을 리용할수 있다. 이 경우에 패턴이 표현하는 파일이름의 긴 목록을 제공하는것보다 인수로서 지령에 이 패턴을 리용할수 있다. 쉘 그자체는 이런 점에서 확장을 수행하며 지령에 확장된 목록을 제공한다.

8.2.1 *와 ?

특수문자들을 보기로 하자. 제6장에서 현재등록부안에 있는 모든 파일들을 지우기 위하여 지령 rm *(6.12)를 사용하였다. **메타문자**(metacharacter)로 알려 진 *는 쉘의 특수문자들중의 하나이다. 이 문자는 일부 몇개의 문자들을 대신한다(없는 문자도 대신한다).

*가 문자렬 chap에 첨부될 때 패턴 chap*는 문자렬 chap로 시작하는 파일이름들(파일 chap를 포함)을 대신한다. 이렇게 앞의 지령행에서 리용된 모든 파일이름들을 대신한다. ls의 인수로서 이 패턴을 리용할수 있다.

```
$ ls -x chap*
chap chap01 chap02 chap03 chap04 chap15 chap16 chap17 chapx chapy
chapz
```

쉘이 이 지령행을 만나면 즉시 메타문자로서 *를 식별한다. 다음 현재등록부로부터 이 패턴과 정합되는 파일목록을 만든다. 그것은 아래에서처럼 지령행을 복구하고 실행하기 위하여 핵심부에 그것을 보낸다.

```
ls -x chap chap01 chap02 chap03 chap04 chap15 chap16 chap17 chapx chapy
chapz
```

인수로서 *를 가지고 echo를 사용할 때 어떤 현상이 일어 나겠는가?

```
$ echo *
array.p vack.sh calendar cent2fah.pl chap chap01 chap02 chap03 chap04 chap15
chap16 chap17 chapx chapy chapz count.pl date_array.pl dept.lst desig.lst
n2words.pl name.pl name2.pl odfile operator.pl profile.sam rdbnew.lst repl.pl
```

간단하게 파일의 목록이 보인다. 쉘은 현재등록부안에 있는 파일들을 정합하기 위하여 *를 사용한다. 모든 파일이름을 정합하므로 출력에서 그것들모두를 볼수 있다.

주해

Windows사용자들은 *가 파일이름안의 아무곳에 놓인다는것을 알면 놀랄수 있다. UNIX에서 *는 끝에만 놓이지는 않는다. *chap*는 다음의 모든 파일이름들 즉 chap newchap chap03 chap03.txt를 대신한다.

또 다른 메타문자는 ?이다. 이것은 하나의 문자를 정합한다. 꼭 같은 chap문자렬에 리용될 때 (chap?로) 쉘은 chap로 시작하는 5개의 문자로 된 파일이름들을 얻어 낸다. 문자렬의 끝에 또 다른 ?를 배치하면 패턴 chap??를 가진다. 이 표현식들을 개별적으로 리용하면 ?의 의미는 명백해 진다.

```
$ ls -x chap?
chapx chapy chapz
$ ls -x chap??
chap01 chap02 chap03 chap04 chap15 chap16 chap17
```

파일이름과 관련된 이 메타문자들을 **통용기호**(wild card:주패에서의 만능패와 같은것)라고도 한다. 쉘의 통용기호의 완전한 목록을 표 8-1에서 실례와 함께 보여 주었다. 거기서 다른 통용기호들의 의미도 보게 될것이다.

표 8-1. 쉘의 통용기호와 응용

통용기호	의 미
*	없는 문자도 포함하여 몇개의 문자들을 정합한다
?	한개 문자를 정합한다
[ijk]	한개 문자 즉 i, j, k를 정합한다
[!ijk]	i, j, k가 아닌 한개의 문자들을 정합한다
[x-z]	x와 z문자의 ASCII범위안에 있는 단일문자를 정합한다
[!x-z]	x와 z문자의 ASCII범위밖에 있는 단일문자를 정합한다

실례

지령	의미
ls *.lst	확장자가 lst인 모든 파일을 보여 준다
mv * ../bin	어미등록부의 bin보조등록부에로 모든 파일을 이동한다
compress .?*.?*	점으로 시작하면서 그뒤에 하나이상의 문자들이 놓이고 두번째 점뒤에 하나이상의 문자들이 놓이는 모든 파일들을 압축한다
cp foo foo*	foo파일을 foo*로 복사한다(여기서 *는 의미를 상실한다)
cp ?????? progs	progs등록부에로 6개 문자로 이름 지어 진 파일들을 모두 복사한다
cmp rep[12]	파일 rep1과 rep2를 비교한다
rm note[0-1][0-9]	note00, note01,..., note19파일들을 제거한다
lp *.[!o]	C목적파일들을 제외하고 확장자를 가진 모든 파일들을 인쇄한다
cp ?*.*[!1238]	점앞에 적어도 한개 문자와 확장자를 가진 파일들을 어미등록부에로 복사한다 그러나 마지막문자가 1, 2, 3, 8인것은 제외한다

주해

패턴을 정합하기 위하여 쉘이 리용하는 통용기호문자들은 정규표현에서의 vi와 emacs가 사용하는것과 비슷하다. 정규식들은 지령(vi와 emacs같은것)에 의해 리해되고 해석되며 쉘에 아무런 작용도 하지 않는다

8.2.2 문자모임

앞의 실례들에서 형성한 패턴들에는 제한이 없었다. 간결한 표현식으로 파일 chapy와 chapz만을 렬거하는것은 쉽지 않다. 또한 수자로 된 목록으로부터 첫 4개의 장들만 따내는것도 역시 쉽지 않다. 이 정합조작을 위해서는 **문자모임**(character class)이 필요하다.

문자모임은 꺾쇠괄호 []안에 표현되는 두개이상의 메타문자들을 리용한다.이 괄호안에 다중문자들을 가질수 있지만 정합은 그 모임안의 개개의 문자에 대해서 진행된다. 실례로 하나의 문자는 표현식에 표현될수 있는 1, 2, 4중의 어느 한 값을 가질수 있다.

```
[124]              1, 2, 4중에 하나
```

이것은 어떤 문자렬이나 또 다른 통용기호표현과 결합될수 있으므로 파일 chap01, chap02와 chap04를 선택하는것은 이제는 간단하다.

```
$ ls -x chap0[124]
chap01 chap02 chap04
```

모임안에 -(이음표)로 범위를 지적할수 있다. [a-h]는 범위를 리용한 문자모임이다. 이것은 파일이름안에 자주 리용되는 문자들이 있기때문에 수자순과 자모순으로 수행된다. 그래서 첫 네개의 수자로 된 장을 선택하려면 범위 [1-4]를 리용하여야 한다.

```
$ ls -x chap0[1-4]
chap01 chap02 chap03 chap04
```

유효범위명세는 왼쪽에 있는 문자가 오른쪽에 있는것보다 더 낮은 ASCII값을 가질것을 요구한다. 이 속성을 리용하면 파일 chapx,chapy와 chapz도 이런 식으로 렬거될수 있다.

```
$ ls -x chap[x-z]
chapx chapy chapz
```

주해

표현식 [a-zA-z]*는 경우에 관계없이 자모순으로 시작되는 모든 파일이름들을 정합한다. [a-zA-Z0-9_]도 물론 수자와 밑선문자를 포함한 단어문자를 정합한다.

8.2.3 문자모임의 부정

!는 통용기호의 모임에서 마지막문자이다. !는 문자모임의 시작에 놓이며 그것의 역할은 정합조건을 반전한다. 즉 그것은 모임안의것을 제외한 다른 모든 문자들을 정합한다. 주해에서 만든 앞의 실례를 반전시키면 그 패턴

```
[!a-zA-z]*              C쉘에서는 동작하지 않을것이다
```

는 첫 문자가 자모가 아닌 모든 파일이름들을 정합한다.

!와 함께 사용된 문자모임은 오직 매개의 문자를 부정하는 수단을 표현한다. 확장자를 가지고 있는 모든 파일(확장자가 .Z인 파일(compress지령으로 압축되였다.)을 제외한)들을 정합하려면 다음의 패턴을 리용할수 있다.

```
*.[!z]             .Z가 아닌 확장자를 가지는 모든 파일들
```

파일묶음의 정보를 구성할 때 하나 또는 최대로 두개의 메타문자패턴들이 그것들모두를 정합할수 있

기때문에 조심해서 파일이름들을 선택해야 한다. 만일 그렇게 하지 않는다면 그것들모두를 호출하는 지령을 사용하여 매번 그것들을 개별적으로 지적하여야 할것이다.

8.2.4 통용기호들이 자기의 의미를 상실했을 때

이제는 모든 통용기호들을 보았으므로 그것들이 패턴안에 놓이는데 따라 서로 다른 의미를 가진다는 것을 알게 되였을것이다. 때때로 통용기호패턴으로 일부 파일이름들을 정합하기 힘든 파일들을 찾을수 있기때문에 그것들을 아는것은 중요하다.

메타문자 *와 ?는 모임내부에서 사용될 때 자기의 의미를 상실하는데 문자그대로 정합되게 된다. 이와 류사하게 -와 !도 모임밖에 놓일 때 자기의 의미를 상실한다. 게다가 !는 모임의 시작위치가 아니라 아무곳에 놓일 때 자기의 의미를 상실한다. -도 또한 량쪽의 문자들이 적당한 경계를 짓지 못한다면 의미를 상실한다.

주해

[!!]는 !가 아닌 한개 문자로 된 파일이름을 정합한다. 이것은 C쉘과 bash쉘에서 동작하지 않는데 !를 다른 목적으로 리용한다. bash는 여기서 [!\!]를 사용할것을 요구하지만 C쉘은 문자모임을 전혀 부정할수 없다.

8.2.5 점의 정합

다른 제한들이 있다. *는 .(점)이나 경로이름의 /으로 시작하는 모든 파일이름들을 정합하지 못한다. 만일 등록부안에 점다음에 적어도 3개 문자를 가진 숨겨 진 파일들을 렬거하고 싶다면 그 점을 명백히 주어야 한다.

```
$ ls -x .???*
.exrc   .news_time   .profile
```

그러나 만일 파일이름이 점을 시작위치가 아니라 아무곳에 포함한다면 명백히 줄 필요는 없다. 실례로 표현식 emp*lst는 파일이름안에 들어 있는 점을 정합한다.

```
$ ls -x emp*lst
emp.lst  emp1.lst  emp22.lst  emp2.lst  empn.lst
```

주해

*는 점으로 시작되는 모든 파일이름을 정합하지 못한다. 이러한 파일이름은 명백히 정합되여야 한다. 하지만 하나의 *는 임의의 개수의 매몰된 점들을 정합할수 있다. 실례로 패턴 fw*92는 파일이름 fwtk2.1.tar.92를 정합한다.

8.2.6 rm을 *와 함께 사용할 때

이 메타문자들은 체계와의 대화속도를 높이는데 도움을 주지만 정확히 지정하지 않으면 위험에 빠질수 있다. 현 단계에서는 주의라는 단어가 적합할것이다. 쉘의 통용기호 특히 *를 사용할 때 모든 chapter들을 제거하는 타자

```
rm chap*
```

대신에 저도 모르게 chap와 *사이에 공백이 들어 갔다면 어쩔바를 몰라 할수 있을것이다.

```
$ rm chap *              아주 위험하다
rm: chap: No such file or directory
```

오유통보문은 여기서 방금 일어 난 위험을 말해 준다. 즉 rm지령은 이 등록부안의 모든 파일들을 제거하였다. rm과 함께 사용된 하나의 *는 쉘이 분리된 인수로 그것을 취급하므로 대단히 위험할수 있다. 이러한 상태에서는 마지막으로 [Enter]건을 누르기전에 지령행을 정지시키고 검사하여야 한다.

만일 쉘이 표현 chap*로 파일을 정합한다면 어떻게 되는가? 쉘은 chap*로 이름 지어 진 파일도 찾는다. 파일이름들을 선택할 때 메타문자들을 사용하는것을 피해야 하지만 만일 한개를 다루어야 한다면 쉘이 그것을 문자그대로 리해하도록 *의 의미를 해제하여야 한다. 이 해제기능은 다음절에서 취급한다.

통용기호들은 쉘의 기능을 표현하며 그것들을 리용하는 지령의 기능은 아니다. 쉘은 chap*가 rm에 대하여 아무것도 의미하지 않기때문에(인수로 파일이름을 사용하는 그 어떤 지령도 같다.) 그 통용기호를 확장하여야 한다. UNIX체계는 쉘이 모든 통용기호표현들을 확장할 때까지 지령의 실행을 막는다.

주해

통용기호가 지령에 아무것도 의미하지 않고 오직 쉘에 대해서만 의미한다고 생각하는것은 전적으로 옳지 않다. find지령(7.15)은 -name열쇠단어에 파라메터로서 통용기호를 접수한다

```
find / -name "*.[hH][tT][mM][lL]" -print          모든 .html과 .HTML 파일들
find . -name "note??" -print                 note후에 2개 문자들
```

여기서 우리는 꼭 같은 통용기호문자들을 사용하고 있지만 이때 그것들은 find지령의 기능이지 쉘의 기능은 아니다. 패턴의 량옆에 인용부호를 줌으로써 우리는 쉘이 이 패턴을 해석조차 할수 없다는것을 확인하였다. 우리는 이 분리기능에 대하여 간단히 배울것이다.

참고

파일을 뜻밖에 삭제하는 위험을 방지하기 위하여서는 언제나 rm -i지령을 호출하도록 rm지령을 전용화하여야 한다. 이것은 별명의 리용(17.4)을 요구하는데 다른 쉘들에 의해서 지원된다. 별명정의는 기동파일(17.9)에 놓일수 있는데 쉘은 사용자가 가입할 때마다 읽는다.

8.3 역사선에 의한 의미해제

일반적으로 파일이름은 쉘메타문자를 포함하지 말아야 한다. 만일 그렇게 하면 무슨 일이 생기는가? >기호에 의하여 chap*로 이름 지어 진 파일을 찾아 보자.

```
$ echo > chap*               만일 동작하지 않으면 chap\*를 사용하시오
$ _
```

빈 프롬프트상태는 파일이 만들어 졌다는것을 암시한다. ls와 함께 리용된 적당한 통용기호패턴은 다음의것을 확인한다.

```
$ ls -x chap*
chap   chap*  chap01  chap02  chap04  chap15  chap16  chap17  chapx
chapy  chapz
```

현재등록부안에 chap*이름을 가진 파일이 있다. 통용기호패턴은 다른것들과 함께 이 파일을 정합한다. 이 파일은 아주 귀찮으며 즉시로 제거되여야 할것이다. 하지만 그것은 쉽지 않을것이다. rm chap*는 그 파일만이 아니라 이 목록안에 있는 모든 파일을 제거하기때문에 리용할수 없다.

그러면 다른 파일들을 지우지 않고 그 파일을 어떻게 제거하는가? 그것이 가능하게 하려면 쉘은 별표를 메타문자로 해석할 대신에 문자그대로 그것을 취급해야 한다. 대답은 또 다른 메타문자인 \ (역사선)으로 그뒤에 놓이는 어떤 메타문자의 의미를 제거하는것이다. *앞에 \ 을 사용하여 그 문제를 해결하시오.

```
$ ls -x chap\*              문자그대로 chap*을 정합한다
chap*
$ rm chap\*
$ ls -x chap\*
chap* not found
```

표현 chap*는 문자그대로 문자렬 chap*를 정합한다. 이것은 쉘이 제공하는 필수적인 기능이며 이 개념이 다른 령역들에도 어떻게 확장될수 있는가를 보게 될것이다. \을 리용하여 어떤 특수한 문자의 기능을 제거하는것을 **의미해제**(escaping) 또는 **전문화해소**(despecializing)라고 부른다.

만일 현재등록부에 파일chap01, chap02, chap03을 가지고 있으며 그다음

```
echo > chap0[1-3]
```

을 사용하여 파일 chap0[1-3]을 만든다면 후에 그 파일에 접근할 때 두개의 꺾쇠괄호의 의미를 해제해야 한다.

```
$ ls -x chap0\[1-3\]
chap0[1-3]
$ rm chap0\[1-3\]              chap0[1-3] 즉 1개 파일을 지운다
$ ls -x chap0\[1-3\]
chap0[1-3] not found           파일을 제거하였다
```

때때로 \문자자체의 의미를 해제하는것이 필요할것이다. 쉘은 이것을 특수한 문자로 취급하므로 그것의 의미를 해제하기 위하여 또 다른 \이 필요하다.

```
$ echo \\
\
$ echo The newline charatr is \\n
The newline character is \n
```

통용기호외에 쉘이 특수하게 취급하는 다른 문자들이 있다. 그것들 대부분은 자주 의미해제를 요구한다. 아래에 그 5개의 특수한 문자들을 보여 준다.

```
$ echo \\\<\>\'\"
\<>'"
```

쉘은 해석작업을 위하여 이 문자들을 사용한다. \,< , >는 입출력지령을 조종하는데 필요하다. '와 "도 특수한 문자들을 보호한다. 우리는 이 장에서 그 문자들을 상세히 보게 될것이다.

[Enter]건의 의미해제

이 통용기호를 제외하고도 쉘에 특별한 다른 문자들 실례로 행바꾸기문자와 같은 문자들이 있다. 지령을 련속적으로 길게 입력하거나 수값인수와 함께 지령을 입력할 때 [Enter]를 누름으로써 지령행을 분할할수 있는데 그것은 이 건의 의미를 해제한후이다.

```
$ find /usr/local/bin /usr/bin -name "*.pl" -mtime +7 -size -1024 \[Enter]
> -size +2048 -atime +25 -print              >을 주목하시오
```

이것은 여러개의 인수들과 함께 자주 리용되는 find지령이다. 의미해제는 긴 지령행들을 읽기 좋게 분할하는 가장 좋은 방법이다. 여기서 \ 은 [Enter]에 의해서 발생되는 행바꾸기문자의 의미를 해제한다. 또한 두번째 프롬프트를 생성하는데(> 혹은 ?일수 있다.) 그것은 지령행이 완성되지 않았다는것을 지적한다.

한개 문자대신에 문자묶음의 의미를 해제해야 하는 경우 역사선에 의한 의미해제(escaping)는 좋은 방법이 못된다. 인용부호화하는것이 더 좋다.

주해

두번째 프롬프트는 >나 ?일수 있는데 그 경우에는 그 지령이 완성된것이 아니라는것을 의미한다. C쉘은 ?를 사용하지만 다른 쉘들은 >를 사용한다.

8.4 인용부호화

특수문자의 의미를 해제하는 또 다른 방법이 있다. 지령인수가 인용부호(quote)로 둘러 막힐 때 그 안의 모든 특수문자들의 의미는 해제된다.

```
$ echo '*?[8-9]'              겹인용부호도 사용할수 있다
*?[8-9]
```

우와 같이 되였을 때 인수는 **인용부호화**(quoting)되였다고 말한다. 겹인용부호도 그러한 목적에 사용되지만 일부 경우에는 일부 특수문자들(특히 $과)에 대한 해석을 진행한다. 초학도들에게 있어서 외인용부호들은 특수문자(인용부호 그자체는 제외하고)들을 보호하기때문에 가장 안전하다.

인용부호화는 공백을 보존한다

공백은 쉘에 있어서 특수한 의미를 가지는 또 하나의 문자이다. 쉘은 지령행에서 련속적인 공백들과 타브들을 발견하면 그것들을 한개의 공백으로 압축시킨다. 다음의 echo지령을 주면 압축된 모든 공백들을 볼수 있다.

```
$ echo The shell          compresses    multiple    spaces
The shell compresses multiple spaces
```

echo의 인수들은 적어도 2개 생길수 있는 공백문자의 의미를 역사선에 의하여 해제하는 방법으로 보존되였다.

```
$ echo The shell \  \  \  copresses \ \  multiple \ \  spaces
The shell       compresses    multiple    spaces
```

사용자가 쉘로부터 보호되여야 하는 많은 문자들을 가지고 있다면 인용부호화하는것이 역사선에 의한 해제보다 오히려 더 낫다.

```
$ echo "The shell           compresses      multiple    spaces"
The shell         compresses    multiple    spaces
```

이때 겹인용부호를 사용하였는데 우에서와 같이 동작하였다. 인용부호들은 또한 \을 보호한다.

```
$ echo '\'
\
```

echo는 드문 지령이다. 지금까지 우리는 쉘이 원래의 의미를 버리도록 하는데 \을 사용하였다. 이 문자는 echo가 서로 다르게 동작하게 하도록 사용될수 있다. 이 기능은 다음절에서 서술한다.

8.5 echo에서의 역사선에 의한 의미해제와 인용부호화

쉘과 별도로 자기 문법의 부분으로서 \을 리용하는 지령들이 있다. \은 특수의미를 제거하는것으로서가 아니라 지령이 그것을 특수문자로 다루도록 그 문자를 강조하는것으로서 리용된다. 실례로 다음의 echo지령을 보자.

```
$ echo 'Enter Your Name : \c'
Enter Your Name : $ _
```

프롬프트는 다음행의 앞부분이 아니라 지령결과문자렬의 끝에 귀환되였다. 쉘은 인용부호때문에 이때에는 \을 해석할수 없다. echo는 \을 해석하고 c문자를 특수문자로 취급한다. 여기서 사용된 \c는 **확장문자렬**(escape sequence)을 표현하는데 유표가 다음행이 아니라 인수다음에 직접 놓이게 한다.

echo는 또한 여러가지 방법으로 유표움직임을 조종하는 다른 확장문자렬을 받아 들인다.

\t- 타브
\f- 페지넘기기
\n- 행바꾸기

아래에 그것들이 사용되는 방식을 보여 준다.

```
$ echo '\tThis message is broken here\n\ninto three lines'
        This message is broken here               타브효과
                                              빈 행으로 통보문을 분할한다
into three lines
```

echo는 또한 인수로서 ASCII 8진수값을 쓴다. 실례로 [ctrl-g]는 삑소리를 낸다. 이 건은 8진수값 007을 가진다. 지령에 인수로서 이 값을 사용할수 있는데 \을 그앞에 놓은후에만 사용할수 있다.

```
$ echo '\007'               겹인용부호도 쓸수 있다
...《삑》소리가 난다...
```

여기서 우리는 UNIX지령에 리용되는 ASCII 8진수값을 처음으로 보았다(극소수 지령들이 8진수를 사용한다). 일부 사람들은 이 문자들의 ASCII값을 사용하여 말단에 칸그리기문자(box-drawing character)들을 표시하기 위해 echo를 사용한다.

BASH쉘

echo에 서술된 확장문자렬은 Linux bash쉘에서는 이런 형식으로 쓰이지 못한다. 그것들을 리용하기 위하여 echo는 -e선택항목도 함께 리용한다.

```
echo -e "Enter your name:\c"
```

우리는 이 책에서 확장문자렬들을 광범히 사용한다. 그러므로 만일 Linux사용자이라면 이 선택항목을 기억해야 한다. 또한 자동적으로 사용된 쉘을 검사하고 이 선택항목을 삽입하는 스크립트(19.13)도 보여 준다.

8.6 방향절환

앞에서 사용한 대부분의 지령들은 말단에 자기의 출력을 보냈다. 또한 건반으로부터의 입력을 가지는 cat(6.14)와 bc(3.12)지령들을 보았다. 이 지령들은 오직 고정된 원천지와 목적지만을 받아 들이도록 설계되였는가? 아니다. 그것들은 사실상 자기의 원천과 목적을 모르고 **문자흐름**(character stream)을 사용하도록 설계되였다. 흐름이란 지령의 입력과 출력으로 되는 바이트들의 렬이다.

UNIX는 이 흐름들을 파일들로 취급하는데 UNIX지령들은 이 파일들로부터 읽고 이 파일들에 쓴다. 지령은 보통 말단에 출력을 보내도록 설계되여 있지 않고 이 파일에 보내도록 설계되여 있다. 마찬가지로 지령은 건반으로부터 입력을 접수하도록 설계되여 있지 않고 오직 흐름으로 되는 표준파일로부터 접수하도록 설계되여 있다. 어떤 프로그람이 발생시킨 모든 오유통보문들을 위한 흐름도 있다. 이 흐름도 파일이다.

쉘은 이 3개의 표준파일들(입력,출력과 오유)을 설치하며 가입하는 때에 사용자말단에 그것들을 첨부시킨다. 흐름들을 사용하는 어떤 프로그람이 이 파일들을 열고 쓸수 있게 한다. 쉘은 또한 사용자가 체계로부터 탈퇴할 때 이 파일들을 닫는다.

입력을 위한 표준파일을 표준입력이라고 하며 출력을 위한 표준파일은 표준출력이라고 한다. 오유흐름은 표준오유라고 한다. 이 표준파일들은 그 자체로는 어떤 물리적인 장치와도 관련되지 않지만 쉘이 그것들에 대하여 기정으로 일부 물리적인 장치를 설정한다.

- 표준입력 : 기정원천지는 건반이다.
- 표준출력 : 기정목적지는 말단이다.
- 표준오유 : 기정목적지는 말단이다.

표준출력과 표준오유는 둘 다 같은 기정장치 즉 말단을 공유하도록 설계되여 있다. 다음의 화제들에서는 쉘이 지령행에 일부 특수한 문자들이 존재하는 경우 이 세 파일들을 디스크의 어떤 물리적인 파일에 재할당(재배치)하는 방법을 보게 될것이다. 이것은 입력은 건반으로부터 들어 오고 출력과 오유는 말단으로 나가는것이 아니라 어떤 디스크파일이나 다른 장치들에서 들어 오거나 나가도록 **방향절환**(redirection)된다는것을 의미한다.

8.6.1 표준출력

cat와 who와 같은 지령들은 자기의 출력을 문자흐름으로서 보낸다. 이 흐름을 지령의 **표준출력**(standard output)이라고 부른다. 즉 기정적으로는 말단에 나타난다. >와 >>기호들을 사용하여 어떤 디스크파일로 출력방향을 바꿀수 있다. 이제 그것을 who지령으로 수행하자.

```
$ who > newfile
$ _                 프롬프트를 돌려 준다
```

쉘은 >을 보고 표준출력이 방향바꾸기되여야 한다고 리해하고 파일 newfile을 열고 그안에 흐름을 쓴 다음에 파일을 닫는다. newfile은 여기서 사용자들의 목록을 가지고 있다(who의 출력). 이것은 우리가 파일안에 지령의 출력을 보존하는 방법이다.

만일 출력파일이 존재하지 않는다면 쉘은 지령을 실행하기전에 출력파일을 만든다. 만일 그것이 존재한다면 쉘은 덧쓰기한다. 따라서 주의해서 이 연산자를 사용해야 한다. 선택적으로 >>(>기호를 두번 사용)기호들을 리용하여 파일에 추가할수 있다.

```
who >> newfile              이미 있는 내용들을 지우지 않는다
```

방향절환은 또한 많은 파일들의 표준출력을 련결할 때 쓸모 있는 기능으로 된다. 통용기호를 사용하여 생략된 지령행을 쓸수 있다.

```
cat chap?? > textbook
```

또한 두개이상의 지령들을 조합할수 있으며 집합된 출력을 어떤 파일로 재배치할수 있다. 한쌍의 괄호는 지령들을 묶어 주며 한개의 >기호는 그것들을 다 재배치하는데 사용될수 있다.

```
( ls -l ; who ) > lsfile
```

이전의 장들에서는 UNIX가 여러가지 형태의 파일들사이에 아주 작은 차이를 준다는것을 완전히 확증할수 없었다. 방향절환은 파일형에 대하여 자주 주의를 돌리지 못한다. 이것은 누군가의 말단에 통보문을 출력하기 위하여 장치이름과 함께 동작할수 있다. 다음의 지령은 말단/dev/tty02에로 통보문을 보내는데 그 말단에서 작업하는 사용자는 그것을 볼수 있을것이다(제공된 말단은 가능하게 된다).

```
echo This message is for the terminal tty02 >/dev/tty02
```

이 말단은 표준출력의 첫번째 목적지이다. 디스크파일은 두번째 목적지이다. 또 다른 프로그람에 입력으로 되는 세번째 목적지가 있는데 이것은 관흐름을 론의할 때 취급한다. 이 3개의 목적지를 리용하여 표준출력흐름을 조종하는것을 그림 8-2에 보여 주었다.

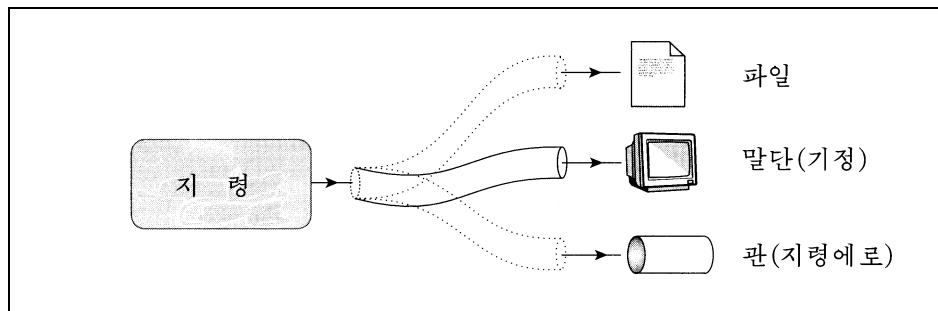



그림 8-2. 표준출력의 3가지 목적지

주해 어떤 지령의 출력이 파일로 보내질 때 출력파일은 지령이 실행되기전에 쉘에 의해서 만들어진다. cat foo > foo는 무엇을 의미하는가?

8.6.2 표준입력

일부 지령들은 자기의 입력도 흐름으로 가지도록 설계되여 있다. 이 흐름을 어떤 지령에 대한 **표준입력**이라고 한다. 1.10.5에서 어떤 파일안에 있는 행, 단어, 문자들을 계수하기 위하여 wc지령을 사용하였다. 하지만 그 지령은 파일이름이 없으면 건반으로부터 입력을 주어야 한다.

```
$ wc                    파일이름이 없다
       2 ^ 32                  표준입력의 시작
      25 * 50                  일부러 제공된 공간
30*25 + 15^2
[Ctrl-d]                표준입력의 끝
       3       9       39              출력안에 파일이름이 없다
```

6.14에서도 이와 류사하게 cat를 사용하였다. 본문(수학식의 모임)의 3개 행들을 입력하고 [Ctrl-d]로 입력의 끌을 지정한 다음 [Enter]를 누른다. wc는 즉시 자기의 표준출력으로 3개 행들, 9개 단어들과 39개의 문자들을 계수한다.

이 입력은 파일로부터 발생하도록 방향을 바꿀수 있다. 먼저 3개의 식으로 calc.lst를 채운다(cat > calc.lst를 사용하여). 이제 메타문자 <가 다음과 같은 식으로 사용될 때 쉘은 wc의 표준입력이 이 파일로부터 들어 오도록 방향을 바꾼다.

```
$ wc < calc.lst
      3       9       39
```

이것 역시 표준입력인데 두번째 형식이다. wc는 파일을 열지 못한다는것을 알아야 한다. 파일이름을 인수로서 사용할 때만 그렇게 할수 있다.

```
$ wc calc.lst
      3       9       39 calc.lst
```

wc는 이때 파일이름을 보여 준다. 즉 wc가 자체로 파일을 열기때문에 아주 잘할수 있다. 만일 지령이 우에서처럼 자체로 파일을 읽을수 있다면 표준입력을 파일로 하는것이 왜 시끄러운가? 대답은 자기입력의 원천지를 모르는 지령을 보관할 필요가 있을 때가 있다는것이다. 체계의 가장 깊이 자리 잡은 기능들중의 하나인 이 측면은 이 장들을 통하여 나가면서 저절로 점차적으로 드러날것이다.

총괄적으로 말해서 표준입력흐름은 3개의 원천지를 가진다.

- 건반(기정원천지)
- <로 방향절환을 리용하는 파일
- 관흐름을 리용하는 또 다른 프로그람(후에 취급하게 될것이다)

이 3가지 원천지들로부터 표준입력흐름을 조종하는것을 그림 8-3에 보여 준다.

주해

표준입력이 파일(<과 함께)로부터 들어 오도록 방향을 바꾸었을 때 파일을 여는것은 쉘이다. 여기서 지령은 총체적으로 쉘의 활동을 모른다. 그러나 파일이름이 인수로 지령에 제공될 때 파일을 여는것은 쉘이 아니라 지령이다.

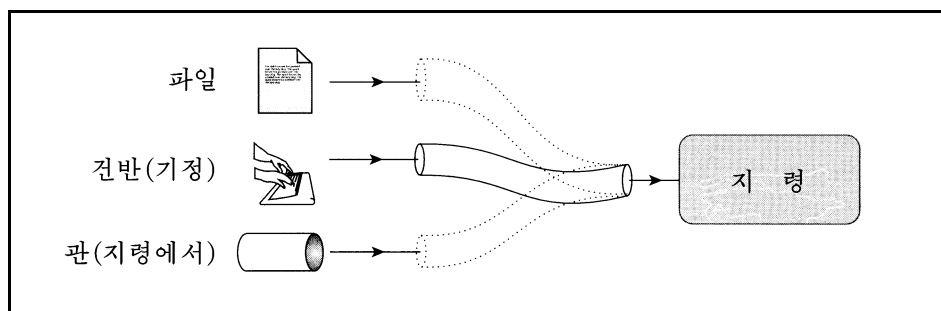



그림 8-3. 표준입력의 3가지 원천지

산수계산을 묶음으로

3.12에서 bc지령을 수산기로 사용하였다. 이 지령에도 몇가지 고찰해야 할 점이 있다. 실례로 일부 산수식들을 포함한 calc.lst파일을 들자. bc의 표준입력방향을 이 파일에로 바꾸자.

```
$ bc < calc.lst > result.lst            표준입력과 표준출력을 둘 다 사용
$ cat result.lst
4294967296              이것은 2^32이다
1250                이것은 25*50이다
975                이것은 30*25 + 15^2이다
```

여기서 어떤 일이 생길수 있는가를 보자. bc는 calc.lst로부터 매행의 계산을 구하고 표준출력상에 결과를 인쇄하였다. 파일안에 그것들을 배치할수 있으며 묶음으로 전체 일감들을 수행한다. 또한 분리된 파일안에 출력을 보존할수 있다(>을 사용). 만일 calc.lst안에 계산된 결과열에 매 표현식을 가질수 있다면 그것이 더 낫다. paste지령(9.11)을 사용하는 방법을 배운후에 그렇게 할것이다.

파일과 표준입력으로부터의 입력

지령이 다중원천지들로부터 즉 파일과 표준입력으로부터의 입력을 가질 때 -기호는 입력을 가지는 렬을 지적하기 위하여 사용된다. 다음렬의 의미는 아주 명백하다.

```
cat - foo              먼저 표준입력으로부터 그리고 다음은 foo로부터
cat foo - bar              먼저 foo로부터 다음은 표준입력으로부터 그리고 다음은 bar로부터
```

여기에 고려하지 못한 표준입력의 4번째 형식이 있다. 쉘프로그람작성의 응용을 보여 주는 here문서가 있는데 후에 제19장에서 론의된다.

8.6.3 표준오유

틀린 지령을 입력하거나 존재하지 않는 파일을 열려고 할 때 어떤 특별한 통보문이 화면상에 나타난다. 이것이 표준오유흐름이다. 표준출력과 같이 역시 말단을 목적지로 한다. 그것들은 사실 두가지로 분리된 흐름들인데 쉘은 그것들을 따로 따로 얻기 위하여 기계적으로 처리한다. 존재하지 않는 파일을 《련결》하려고 하면 세번째 흐름을 발생시킨다.

```
$ cat bar
cat: cannot open bar: No such file or directory
```

표준오유흐름은 또한 파일에 재할당될수 있다. 분명히 표준출력을 위한 기호를 사용하면 안된다.

```
$ cat bar > errorfile
cat: cannot open bar: No such file or directory
```

사용자가 존재하지 않는 파일을 《련결》하기 위해 노력하였지만 오유통보문에 여전히 말단에 나타난다. 우리는 더 나가기전에 세개의 모든 표준파일들이 **파일서술자**(file descriptor)라고 부르는 번호를 가진다는것을 알아야 하는데 이 서술자는 식별에 리용된다.

0-표준입력   <는 0<와 같다.
1-표준출력   >는 1>와 같다.
2-표준오유   오직 2>이여야 한다.

이 서술자들은 방향절환기호들의 앞에 기정적으로 붙여 진다. 실례로 >와 1>는 쉘에 있어서 같은것을 의미하며 동시에 <와 0<도 같다. 번호 0과 1들은 기정값들이므로 일반적으로 방향절환기호들앞에 앞붙이로 붙일 필요는 없다. 그러나 표준오유를 위한 서술자 2>는 사용할 필요가 있다.

```
$ cat bar 2>errorfile
$ cat errorfile
cat: cannot open bar: No such file or directory
```

또한 표준출력을 추가하는것과 류사한 방법으로 아래와 같이 특별한 출력을 추가할수 있다.

```
cat bar 2>> errorfile
```

우와 같은 방법으로 개별적인 파일안에 오유통보문들을 보존할수 있다. 즉 이것은 긴 프로그람들을 실행하여 그날의 마지막에 보게 될 오유출력을 보존해 둔다.

C쉘

표준오유는 C쉘에 의해서 서로 다르게 처리되므로 이 절의 실례들은 그것으로는 동작하지 못할것이다. 사실상 C쉘은 표준출력과 표준오유를 통합한다. 즉 표준오유만을 조종하는 기호는 따로 없다.

8.6.4 흐름의 결합

지령이 파일이름 혹은 -없이 사용되면 그것은 표준입력으로부터 입력된다는것을 의미한다. 출력 역시 그렇게 정할수 있다. 이 두개의 형식들은 같다.

```
cat > foo            -는 여기서 필요 없다
cat - > foo           입력과 출력이 둘 다 정해 졌다
```

6.14에서는 우의 첫번째 지령을 사용하여 입력과 출력의 방향을 바꾸어 파일을 만들었다. 또한 <와 >연산자들을 결합할수 있다. 즉 그것들의 렬에는 제한이 없다.

```
wc < infile > newfile          먼저 입력, 다음에 출력
wc>newfile<infile            먼저 출력, 다음에 입력
> newfile < infile wc          우에서와 같지만 끝에서 지령을 입력
```

<, >, >>연산자들은 자기 주위에 있는 공백들에 무관계하다. 이 모든 경우에 쉘은 원천지와 목적지를 둘 다 모르는 지령을 보존한다. 마지막실례에서 wc는 지령행에서 마지막단어이다.

표준출력과 표준오유기호들은 또한 같은 지령행에서 사용될수 있다.

```
cat newfile nofile 2> errorfile > outfile          그러나 결코 같은 파일이 아니다
```

때때로 같은 파일안에 표준출력과 표준오유흐름들을 둘 다 쓸 필요가 있을것이다. 그러면 일부 특수한 기호들을 사용해야 하는데 제19장에서 그것들에 대하여 배운다.

8.6.5 지령들의 새로운 분류

모든 지령들이 표준입력과 표준출력의 기능을 사용하는가? 그렇지 않다. 이러한 견지로부터 UNIX지령들은 4가지 부류로 분류될수 있다.

지 령	표준입력	표준출력
mkdir,rmdir,cp,rm	없다	없다
ls,pwd,who	없다	있다
lp,lpr	있다	없다
cat,bc,wc	있다	있다

입력의 원천지와 출력의 목적지에 대한 지령의 무관심성은 UNIX체계의 가장 훌륭한 기능의 하나이다. 이것은 한 지령의 출력이 또 다른 지령의 입력으로 될수 있기때문에 지령들이 또 다른 지령과 《대화》할수 있는 가능성을 준다. 이 통신들을 허락하는 관흐름(pipe line)에 대해서는 후에 취급한다. 3가지 흐름의 조종을 그림 8-4에 보여 준다.

8.7 특수파일 /dev/null과 /dev/tty

흔히 화면상에서 출력 또는 오유통보문을 보지도 않고 프로그람이 성과적으로 실행되는지 안되는지 시험하고 싶을수 있다. 이 출력을 파일들안에 보관하고 싶지 않을수도 있다. 크기를 변화시키지 않고 어떤 흐름을 간단히 받아 들이는 특수한 파일 /dev/null이 있다.

```
$ cal 1995 >/dev/null
$ cat /dev/null          크기는 항상 0이다
$ _
```

장치파일 /dev/null은 간단히 그쪽으로 향한 모든 출력을 소각해 버린다.그것의 크기는 항상 0이다. 이 기능은 오유통보문들이 화면상에 나타나지 않도록 그 방향을 바꾸어 주는데 쓸모 있다.다음의 렬은 현시를 하지 않고 존재하지 않는 파일을 《련결》하려고 한다.

```
cat chap100 2>/dev/null          표준오유의 방향을 바꾼다
```

/dev/null은 사실 모조장치이므로 다른 모든 장치파일들과 달리 어떤 물리적인 장치와 련결되여 있지 않다.

UNIX체계에서의 또 다른 특수파일은 어떤 말단을 지적하는 파일 즉 /dev/tty이다. 실례로 romeo가 /dev/tty01말단에서 작업하고 juliet가 /dev/tty02에서 작업하고 있다고 생각하자. 그러나 romeo와 juliet는 둘 다 자기의 말단들을 한개의 장치파일 즉 /dev/tty로 여긴다. 만일 romeo가 지령

```
who >/dev/tty
```

를 주면 현재사용자들의 목록은 그가 현재 리용하고 있는 말단 /dev/tty01로 보내진다. 류사하게 juliet도 자기 말단 /dev/tty02상에서 출력을 보기 위해 같은 지령들을 사용할수 있다. /dev/null과 같이 /dev/tty 는 여러 사용자들이 충돌이 없이 독립적으로 접근할수 있는 또 다른 특수파일이다.

출력이 그 말단에 가도록 기정적으로 되여 있는데도 왜 어떤 지령의 출력에 대해서는 그 말단으로 가도록 방향을 정해 주어야 하는가? 때때로 그것을 꼭 지적하는것이 필요하다. 이 파일은 방향절환에 리용되는것외에도 일부 UNIX지령들에 인수로서 리용될수 있다. 8.9에서 이 기능을 리용하는데 일부 상황들은 제19장(쉘프로그람작성기능)에서 취급된다.

참고

만일 뿌리등록부로부터 탐색을 시작하기 위하여 권한이 없는 보통의 등록자리로 find를 리용한다면 지령은 등록부에 "cd"가 불가능하다는 오유통보문을 발생시킬것이다. 오유가 발생된 목록에서는 선택한 파일을 놓칠수 있으므로 find의 표준오유는 find/-name typescript-print 2>/dev/null과 같이 /dev/null에 지적되여야 할것이다. C쉘에서는 이렇게 할수 없다.

8.8 관과 지령사이의 련결

관(pipe)을 리해하기 위하여 현재 가입된 사용자들의 수를 계수하는 과제를 수행해 보자. 먼저 우리가 이미 소유한 지식을 사용하여 그 과제를 수행해 본다. who는 행당 한명의 사용자가 놓인 사용자들의 목록을 현시하는데 우리는 이 출력을 한개 파일안에 보관하려고 한다.

```
$ who > user.lst
$ cat user.lst
romeo       tty01       May 18 09:32
juliet      tty02       May 18 11:18
andrew      tty03       May 18 13:21
```

이제 user.lst로부터 들어 오도록 wc -l지령의 표준입력방향을 바꾸면 사용자들의 수는 실제적으로 계수된다.

```
$ wc -l < user.lst
       3              사용자들의 수
```

2개의 지령들을 차례로 사용하는 방법은 다음과 같은 결함을 가지고 있다.

- 처리속도가 뜨다. 두번째 지령은 첫번째 지령이 그 일감을 완성할 때까지 동작할수 없다.
- wc지령이 자기 실행을 끝낸후에 제거되여야 하는 중간파일이 필요하다.
- 큰 파일을 처리할 때 림시파일들이 쉽게 생길수 있으며 그러면 디스크공간이 랑비된다.

여기서 who의 표준출력방향을 바꾸었으며 따라서 그것은 wc의 표준입력으로 되였다. 여기서 의문이 있을수 있다. 쉘은 어떤 지령이 다른 곳으로부터의 입력을 가지도록 이 흐름들을 하나로 련결할수 없는가? 그렇게 할수 있다. 쉘은 두 지령의 련결자로서 특별한 연산자 |를 리용한다. who와 wc작업을 하나로 만들수 있다. 어떤 지령이 다른 곳으로부터의 입력을 가지도록 한다.

```
$ who | wc -l
       3
```

여기서 who는 wc에 관련결된다(piped)고 말한다. 이것을 리용하면 중간파일이 만들어 지지 않는다. 지령들이 이런 식으로 하나로 결합될 때 **관흐름**(pipeline)이 형성되였다고 말한다. 이 이름은 관을 련결하는것과 류사하게 프로그람들사이에 그것을 련결하므로 적합하다. 이러한 호상련결을 설치하는것은 쉘이며 지령은 그에 대하여 아무것도 모른다.

관은 각각 표준입력과 표준출력의 세번째 원천지와 목적지이다. 이제는 현재등록부안에 있는 파일들을 계수하기 위한 어떤 지령을 사용할수 있다.

```
$ ls | wc -l
   15
```

비록 설계자들이 이 조작을 수행하기 위하여 ls에 또 다른 선택항목을 쉽게 제공할수 있더라도 개별적인 지령들에는 설계되여 있지 않다. 그리고 wc는 표준출력을 사용하기때문에 사용자는 이 출력이 어떤 파일에로 향하도록 방향을 다시 지정할수 있다.

```
ls | wc -l >fkount
```

관흐름에 사용할수 있는 지령의 수에는 제한이 없다. 그러나 거기에 그것들을 배치하자면 이 지령들의 동작속성들을 알아야 한다. 이 일반적인 지령행을 보자.

```
지령1 |지령2 |지령3 |지령4
```

지령2와 지령3이 표준입력과 표준출력을 둘 다 제공해야 한다는것이 아주 명백하다. 지령1은 표준출력만을 사용하는것이 필요하며 지령4는 표준입력으로부터 읽을수 있어야 한다. 만일 그것을 확신할수 있다면 그림 8-4에서 보여 주는것처럼 하나로 련결된 도구들을 가질수 있다. 두 흐름들을 다 제공하는 지령2와 지령3을 **려파기**(filter)라고 부른다. 려파기들은 도구묶음의 중심에 있는 도구인데 후에 4개의 장들에서 론의된다.

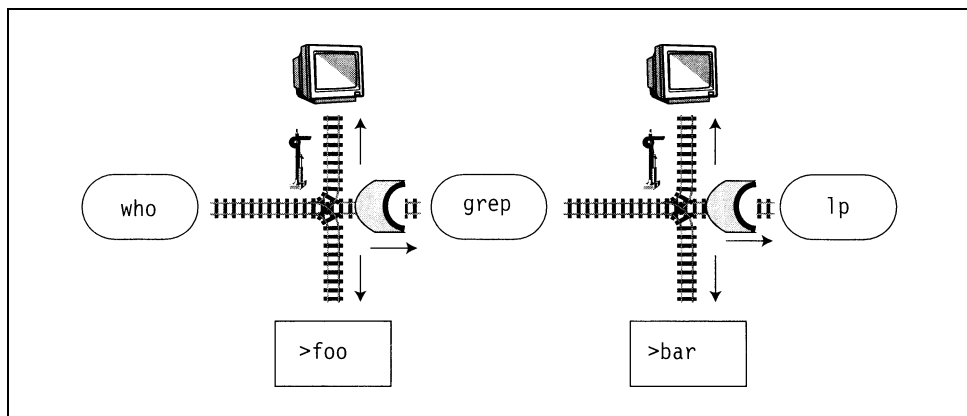



그림 8-4. 세 지령들의 관흐름

man페지들을 인쇄하기

지령의 직결man페지들은 흔히 굵은체로 강조된 열쇠단어들을 보여 준다. 이 페지들은 그것들을 인쇄하기전에 제거되여야 하는 많은 조종문자들을 포함한다. col -b지령은 그것의 입력으로부터 이 문자들을 제거할수 있는데 이것은 man출력이 col -b에 관련결되여야 한다(piped)는것을 의미한다.

```
man grep | col -b > grep.txt
```

이 지령은 본문파일에 평문(clear text)을 보내지만 우리는 그 페지를 인쇄하기 위하여 그것을 다시 관련결할수 있다. lp지령(6.16)은 파일을 인쇄하지만 표준입력을 허락한다.

```
man grep | col -b | lp
```

지령이 자기의 자원을 무시하여야 할 필요가 있을 때

앞에서 본 방향절환(redirection)과 관련결(piping)에 대한 모든 론의로부터 무엇을 알수 있는가? 인수로서 파일이름을 사용하는 grep지령을 생각해 보자.

```
$ grep "print" foo1              grep는 foo1파일을 연다
print "Content-type: text/html\n\n";
print "</html>\n";
```

grep(15.2)는 입력에 지적된 패턴을 포함하고 있는 행들의 위치를 알아 낸다. 여기서는 print문자렬을 포함하고 있는 행들을 현시한다. grep는 또한 려파기이기도 하므로 표준입력으로부터의 입력을 가질

수 있다.

```
grep "print" < foo1              shell은 파일 foo1을 연다
```

이것도 같은 출력을 생성한다. 그러면 실지로 어떤 차이가 있는가? 이 질문은 8.6.2에서 제기되였지만 우리는 이번에 그것을 대답해야 할것이다. 왜 grep가 때때로 파일보다 오히려 흐름을 조종하도록 하는것이 필요한가를 알기 위하여 grep가 다중파일이름도 접수한다고 생각하자.

```
$ grep "print" fool foo2 foo3
foo1:print "Content-type: text/html\n\n";
fool:print "</html>\n";
foo2:find / -mtime +7 -print | perl -ne 'chop ; unlink ;'
foo3:$slno++ ; print ($slno . " " . $_ . > "\n") ;
foo3:printf "File $file was last modified %0.3f days back \n",$m_age ;
```

grep는 이때 파일이름들을 인쇄한다. grep는 파일을 열고 그것들의 이름을 알기때문에 이 과제를 수행할수 있다. 그러나 때때로 파일이름을 제거하고 오직 내용에만 흥미를 가질수 있을것이다. 만일 grep가 자기 입력의 원천지를 무시하도록 만든다면 그렇게 할수 있다. cat로 파일들을 련결하고 grep에 결합된 출력을 관련결한다.

```
$ cat foo[123] | grep "print"
print "Content-type: text/html\n\n";
print "</html>\n";
find / -mtime +7 -print | perl -ne 'chop ; unlink :'
$slno++ ; print ($slno . " " . $_ . "\n");
printf "File $file was last modified %0.3f days back \n", $m_age ;
```

이때 grep가 흐름상에서 동작하므로 파일이름들은 출력에서 없어 진다.

주해

관흐름에서 \의 왼쪽에 있는 지령은 표준출력을 사용하여야 하며 오른쪽에 있는 지령은 표준입력을 사용하여야 한다.

8.9 흐름을 가르기(tee)

UNIX의 tee지령은 자기의 입력을 2개의 구성요소로 가른다. 즉 한 구성요소는 어떤 파일에 보존되며 다른것은 표준출력과 련결된다. tee는 자기의 입력에 대해서 어떠한 려과동작을 수행하지 못한다. 즉 자기가 가지고 있는것을 그대로 내보낸다. 사실 이 지령은 쉘의 기능이 아니지만 문자흐름의 조종을 동반하기때문에 이 장에 포함시켰다.

또한 려과기(표준입력과 표준출력을 사용한다.)가 있으므로 tee는 관흐름의 아무곳에 놓일수 있다. tee지령을 리용하여 who지령의 출력을 파일에 보존할수 있으며 그것을 현시할수도 있다.

```
$ who | tee user.lst
romeo      tty01    May 18 09:32
juliet     tty02    May 18 11:18
```

```
andrew     tty03    May 18 13:21
```

이 현시를 user.lst파일의 내용과 비교검사할수 있다.

```
$ cat user.lst
romeo     tty01    May 18 09:32
juliet    tty02    May 18 11:18
andrew    tty03    May 18 13:21
```

tee의 출력을 또 다른 지령(실례로 wc)에 관련결할수 있다.

```
$ who | tee user.lst | wc -l
       3
```

말단에 사용자의 목록과 그 수를 둘 다 현시하려면 tee를 어떻게 사용해야 하는가? 말단은 또 하나의 파일이기도 하므로 tee에 인수로서 장치이름 /dev/tty를 사용할수 있다.

```
$ who | tee /dev/tty | wc -l              /dev/tty는 지령인수로 사용하였다
romeo     tty01     May 18 09:32
juliet    tty02     May 18 11:18
andrew    tty03     May 18 13:21
     3
```

말단을 파일처럼 취급하는것이 가지는 이 유리한 측면은 우의 실례로부터 명백하다. 만일 tee가(혹은 임의의 UNIX지령) 조종할수 있는 파일형에 제한을 준다면 그렇게 하지 못할수 있다. 여기서 말단은 어떤 디스크파일과 같은 방법으로 취급된다. tee는 또한 출력을 덧쓰기하는것보다 추가하는 -a(append) 선택항목을 사용한다.

8.10 지령대입

쉘은 또 다른 방법으로 2개의 지령들을 련결할수 있게 한다. 쉘은 관(pipe)을 리용하여 어떤 지령의 표준출력을 다른 지령의 표준입력에 련결할뿐아니라 지령의 인수를 다른 지령의 표준출력에서 취할수 있게 한다. 이 기능을 **지령대입**(command substitution)이라고 부른다.

간단한 실례를 들기 위하여 다음과 같은 형태로 날자를 현시하려고 한다고 하자.

```
The date today is Wed Oct 20 10:19 EST 1999
```

이 서술문의 마지막부분(《Wed》로부터 시작)은 date지령의 출력을 표현한다. date지령을 echo지령에 어떻게 편입시키겠는가? 그것은 지령대입에 의하여 해결될수 있는 간단한 문제이다. echo에 인수로 표현식 `date`를 사용하시오.

```
$ echo The date today is `date`
The date today is Wed Oct 20 10:12:19 EST 1999
```

지령행에서 `(역인용부호)는 쉘에서 쓰이는 또 다른 메타문자이다. 건반에 이 문자를 발생하는 특수한 건(일반적으로 왼쪽끝대기에)이 있는데 외인용부호(')와 혼돈하지 말아야 한다. 쉘은 그안의 지령을 실행하고 둘러 막힌 지령행을 그 지령의 출력으로 교체한다. 지령대입을 위해서 《역인용부호화》된 지령

은 표준출력을 리용하여야 한다. 즉 date는 지령대입동작을 한다.

쓸모 있는 통보문들을 발생시키도록 이 기능을 사용할수 있다. 실례로 관호름에 2개의 지령을 사용할수 있는데 그다음 세번째 자리에 인수로서 출력을 사용한다.

```
$ echo "There are `ls | wc -l` files in the current directory"
There are 58 files in the current directory
```

지령은 비록 인수들이 겹인용부호화되였다 하더라도 적당히 동작하였다. 그것은 외인용부호를 사용할 때와 전혀 다르다.

```
$ echo 'There are `ls | wc -l` files in the current directory'
There are `ls | wc -l` files in the current directory
```

이것을 통하여 외인용부호와 겹인용부호사용의 차이점을 한가지 알수 있다. `는 겹인용부호들안에 놓일 때 쉘에 의해서 해석되는 문자들중의 하나이다. 만일 `를 문자그대로 출력하고 싶다면 외인용부호를 사용해야 한다.

지령대입은 흥미 있는 응용가능성들을 가지고 있다. 그것은 한개의 명령문으로 몇개의 명령들을 련결함으로써 작업속도를 높이게 한다. 다음장들에서 이 기능에 대하여 더 볼것이다.

주해

지령대입은 역인용부호와 그안의 지령이 겹인용부호안에 놓일 때 가능하게 된다. 만일 외인용부호를 사용하면 불가능하다.

Korn쉘

Korn쉘과 bash쉘도 역인용부호에 해당한 동의어들을 제공한다. 괄호안에 지령을 놓고 그 문자렬앞에 $을 놓을수 있다.

```
$ echo $(date)
Mon Sep 20 20:09:23 EST 1999
```

BASH쉘

만일 이러한 쉘들중에 어느 하나를 사용하고 있다면 읽을수 없을뿐아니라 낡은 역인용부호를 사용하는 형식보다는 오히려 이 양식을 적용할것이다. 이것은 물론 POSIX가 권고하는 양식이다.

8.11 쉘변수

지령행과 쉘스크립트안에 변수들을 정의하고 사용할수 있다. 이 변수들을 **쉘변수**라고 부른다. 쉘변수는 문자렬형태로서 값이 2진형식이 아니라 ASCII로 보관된다는것을 의미한다. 쉘변수를 사용하기전에 형선언은 필요 없다.

모든 쉘변수들은 일반화된 형식인 변수=값(C쉘에서는 제외)을 가진다. 그것들은 =연산자와 함께 할당되지만 변수이름앞에 $을 붙여 값을 얻는다. 여기에 실례가 있다.

```
$ x=37              =의 량쪽에 공백이 없다
$ echo $x           $은 값을 얻을 때에만 필요하다
37
```

변수이름은 자모, 수자, 밑선문자들을 포함할수 있다. 그러나 첫번째 문자는 자모여야 한다. 그리고 쉘은 대소문자를 구별한다. 즉 변수 x는 X와 다르다. 변수를 제거하려면 unset를 사용하시오.

```
$ unset x
$ echo $x
                              변수가 제거되였다
$ _
```

모든 쉘변수들은 기정적으로 빈 문자렬로 초기화되여 있다. 때때로 그것들을 빈 문자렬로 설정할 필요가 있을것이다.

```
x=              x=''              x=""
```

주의

쉘변수들에 값을 할당하려면 =의 량쪽에 공백이 없는가 확인하시오. 만일 공백들을 삽입한다면 쉘은 변수를 지령으로 취급할것이며 =와 값은 인수로 취급할것이다. 이 제한은 C쉘에서는 적용되지 않는다.

변수에 여러개 단어들로 된 문자렬들을 할당하기 위하여 공백문자의 의미를 해제(역사선에 의하여) 할수 있지만 인용부호화하는것이 더 낫다.

```
$ msg='You have mail' ; echo $msg
You have mail
```

이제 $와 함께 주어 지는 또 다른 특수문자들이 있는데 그것은 쉘에 의해 처리된다. 아직은 값을 구하지 않고 문자그대로 해석하는것이 필요할수 있다. 이것은 $를 포함하고 있는 표현식을 외인용부호로 막거나 $에 역사선(\)을 주어 수행될수 있다.

```
$ echo 'The average pay is $1000'
The average pay is $1000
$ ech The average pay is \$1000
The average pay is $1000
```

출력은 비록 예측할수 있지만 겹인용부호안에 인수들이 놓이면 다른 결과를 얻는다.

```
$ echo "The average pay is $ 1000"
The average pay is 000
```

여기에 외인용부호와 겹인용부호리용의 두번째 차이점이 있다. 역인용부호와 같이 $는 그것이 겹인용부호화될 때에도 쉘에 의해서 평가된다. 여기서 쉘은 "변수"$1의 값을 구했다. 즉 그것은 정의되지 않았으므로 빈 문자렬이 출력되였다. $1은 사용자가 스크립트에 보내는 인수들을 의미하는 위치파라메터(positional parameter)(18.4)라고 부르는 파라메터들의 모임에 속해 있다.

주해

지령대입과 마찬가지로 변수의 평가는 외인용부호안에서 진행되지 못하고 겹인용부호안에서만 진행된다.

C쉘

C쉘은 변수들을 설정하는데 set명령문을 사용한다. =의 량쪽에는 공백이 있어야 하든가 전혀 없어야 한다.

```
set x = 10
set mydir=`pwd`
```

변수평가는 변수이름앞에 $을 붙이는 일반적인 방식으로 진행된다. C쉘은 변수의 다른 형을 설정하는 또 다른 명령문 setenv를 사용한다. 제17장에서 set와 setenv에 대해서 서술한다.

변수들의 사용

변수에 경로이름 설정하기

지령행에서 변수에 경로이름을 설정할수 있으며 그다음 cd지령과 함께 간략표현을 리용한다.

```
$ mfile='/usr/spool/mail'
$ cd $mfile
$ pwd
/usr/spool/mail
```

이제 스크립트안에서 절대경로이름(/usr/spool/mail)을 여러번 사용해야 한다고 가정하자. 스크립트의 시작에서 변수에 경로이름을 할당하고 그다음 그것을 모든 곳 즉 이 스크립트에 의해 실행되는 다른 스크립트에서라도 사용할수 있다.

후에 우편등록부의 위치를 /var/spool/mail로 변화시킬수 있다. 모든 작업이 이전대로 수행되기때문에 변수정의를 다른것으로 변화시킬 필요가 있다.

변수에 UNIX지령을 설정하기

쉘변수는 지령 그자체를 교체하는데 사용될수 있다. 만일 매번 특수한 선택항목들과 함께 지령을 사용하고 있다면 지령행을 변수에 설정한다. 그것을 실행하기 위하여 $을 변수의 앞에 붙인다.

```
$ backup="tar -cvf /dev/fd0h1440 *"          fd0은 장치이름이다
$ $backup            변수가 실행되는 방법을 보여 준다
....
```

이제 보게 될 출력은 파일을 여벌복사하는데 리용되는 UNIX편의프로그람인 tar지령의 실행결과이다. 여기서 또다시 변수를 정의하고 그것을 아무데서나 사용하는 유리한 점을 알수 있다. 만일 여벌장치를 변화시킨다면 정의에서 새 장치이름을 가진 fd0h1440으로 교체한다.

지령대입기능을 리용하여 변수를 설정하기

지령대입기능을 리용하여 변수를 설정할수 있다. 실례로 만일 변수 mydir에 현재등록부의 완전한 경로이름을 설정하려고 한다면

```
$ mydir=`pwd`
$ echo $mydir
/home/romeo
```

를 사용할수 있을것이다. 변수사용은 사용자에게만 제한되지 않는다. UNIX체계도 동작을 조종하는 일부 변수들을 사용한다. 사용하고 있는 말단의 형, 리용하는 프롬프트문자렬, 혹은 들어 오는 우편들을 보관하는 등록부를 말해 주는 변수들이 있다. 이 변수들은 많은 방법으로 그 환경의 조작을 변경할수 있기때문에 **환경변수**(environment variables)라고 부른다. 이 특수한 쉘변수들의 의미에 대한 구체적인 론의는 제17장에서 취급될것이다.

8.12 쉘스크립트

쉘은 어떤 파일안에 지령묶음을 보관하고 그 파일을 실행시키는 기능을 제공한다. 이러한 모든 파일들을 **쉘스크립트**(shell script)라고 부른다. 쉘스크립트를 쉘프로그람, 쉘수속(shell procedure)이라고도 한다. 이 파일들에 보관된 명령들은 해석방식 즉 Windows의 묶음(.BAT)파일과 같이 실행된다. 다음의 쉘스크립트는 script.sh파일안에 보관된 3개의 지령렬을 가지고 있다. vi나 emacs로 파일을 만들수 있지만 이것은 오직 3개 행만 가지므로 대신 cat를 사용할수 있다.

```
$ cat > script.sh
directory =`pwd`              표준입력의 시작
echo The date today is `date`
echo The current directory is $directory
[Ctrl-d]              표준입력의 끝
$ _
```

확장자 sh는 오직 식별목적을 위하여 리용된다. 즉 그것은 아무러한 확장자를 가질수 있으며 지어 없을수도 있다. 간단히 파일이름을 호출함으로써 다음의 지령들을 포함하고 있는 파일을 실행해 보시오.

```
$ script.sh
script.sh: execute permission denied
```

실행허가권은 보통 어떤 쉘수속을 실행하는데 반드시 필요하지만 기정적으로 파일은 만들어 질 때 이 허가권을 가지고 있지 못한다. 먼저 그것을 실행하기전에 파일에 실행가능한 상태를 만들기 위하여 chmod를 사용하시오.

```
$ chmod u+x script.sh
$ script.sh
The date today is Thu Feb 17 11:30:53 EST 2000
The current directory is /home/sumit/project5
```

스크립트는 순서대로 3개의 명령문들을 실행시킨다. 비록 우리가 쉘을 해석기로 사용하였다 해도 그것은 또한 프로그람작성언어이다. 쉘스크립트안에 if, while, for와 같은 모든 표준구조들을 가질수 있다. UNIX체계의 동작은 체계가 기동되는 동안 실행되며 체계관리자에 의해 미리 씌여 진 많은 스크립트들에 의해 조종된다. 이 책에 있는 2개의 장들은 쉘프로그람작성을 론의한다.

8.13 쉘의 지령행처리

지금까지 쉘의 기본기능들을 보았는데 이제는 지령을 처리할 때 뒤따르는 여러개의 단계를 리해하자. 지령행이 [Enter]에 의해 완료된후 쉘은 지령행을 처리하여 나간다.

그 순서는 다음과 같이 정의되여 있다.

- **구문분석**: 쉘은 먼저 인용부호가 없으면 공백과 타브를 리용하여 지령행을 단어들로 가른다. 공백이나 타브가 련속적으로 발생한것들은 여기서 한개의 공백으로 교체된다.
- **변수평가**: $가 앞에 붙어 있는 모든 단어들은 인용부호가 없거나 역사선이 없으면 변수로

평가된다.

- **지령대입**: 역인용부호로 둘러 막힌 지령은 쉘에 의해 실행되는데 그의 출력은 그 지령을 발견한 장소에 삽입된다.
- **방향절환**: 쉘은 그다음 그것들이 지적한 파일을 열기 위하여 >, <와 >>문자들을 찾는다.
- **통용기호해석**: 쉘은 통용기호가 있는가를 보기 위하여 지령행을 훑어 보고 그것들을 패턴에 맞는 파일이름목록과 교체한다.
- **경로평가**: 마지막으로 그 지령을 요구하는 순서대로 탐색하여야 할 등록부들을 결정하기 위하여 PATH변수를 찾는다.

앞의 순서는 Bourne쉘의 거동패턴의 간단한 처리법으로 생각할수 있다. 즉 C쉘은 서로 다른 패턴을 가지고 있다. 실례로 문자 ;(||과 &&도)은 쉘이 더 읽는것을 멈춘다.

8.14 다른 쉘

초기의 UNIX체계는 Bourne쉘과 함께 발전하여 왔다. 그다음 새로운 기능을 제공하는 많은 쉘들이 나왔는데 그중 2개 즉 Korn쉘(지령파일 ksh로 표현)과 bash쉘(지령파일 bash로 표현)은 UNIX애호가들이 적극 받아 들이였다. Korn쉘은 SVR4에서 표준으로 제공되고 bash는 Linux에서 표준쉘이다. C쉘(csh)은 그것들보다 먼저 나왔지만 Korn과 bash에 밀려 나지 않고 여전히 광범히 쓰이고 있다.

Korn과 bash는 Bourne쉘과 서로 밀접히 련관되여 있는데 이것은 Bourne쉘밑에서 개발된 모든 쉘프로그람들이 이 두개의 쉘들밑에서도 실행될수 있다는것을 의미한다. kill과 같은 지령들과 옹근수문자렬처리기능들이 내장되여 있기때문에 Korn과 bash밑에서 실행되는 프로그람들은 Bourne밑에서 보다 더 빨리 실행한다. 이 책에서는 C, Korn, bash쉘들의 기능들이 적당히 강조되여 있다. Korn과 bash의 상반되는 프로그람작성기능들은 제19장에서 론의된다. C쉘의 프로그람작성구조들은 부록 1에 문서화되여 있다. 모든 4개의 쉘들에 대하여서는 부록 4에서 서술한다.

8.15 Korn쉘과 bash쉘에서의 다른 통용기호들

흔히 파일이름의 묶음을 정합하는 어떤 단일한 표현을 형성할수 없는 경우가 제기될수 있다. 실례로 다음의 파일이름들을 어떻게 정합하겠는가?

```
chap01 chap02 chap03 chap16 chap17 chap18 chap19
```

Bourne쉘은 우의 매 파일이름안에 공통문자렬(chap)이 있다 해도 2개의 표현 즉 chap0[1-3] chap1[6-9]를 사용하기만 한다. bash와 Korn은 대괄호안에 서로 다른 표현식들을 넣음으로써 표준통용기호모임에 중요한 확장을 제공한다. 이 표현은 우의 파일들 모두를 정합한다.

```
chap{0[1-3],1[6-9]}                반점을 주의하시오
```

여기서 반점은 괄호안에 놓인 서로 다른 표현사이의 구분문자로 동작한다. 그 량쪽에 공백문자들이 있어서는 안된다. 그리고 아래에 README와 INSTALL파일의 .txt와 .gz판본을 복사할수 있는 방법을 보여 준다.

```
cp {INSTALL,README}.{gz,txt} ../doc
```

이 기능은 지령행을 현저히 줄인다. Bourne에서는 모든 4개의 파일이름들을 개별적으로 지적하여야 하였다. 우의 사실은 생략된 문법을 사용하여 다중등록부들에 접근할수 있다는것을 의미한다.

```
cp /home/romeo/{project,html,scripts}/* .
```

이것은 3개의 등록부(project, html, scripts)로부터 현재등록부까지의 모든 파일들을 복사한다. 이것은 확실히 편리한 기능이다. 이러한 기능은 C쉘에서도 쓰이는데 여기서는 더 론의하지 않는다.

반대선택기능

만일 Windows Explorer를 사용하였다면 의심할바없이 반대선택(Invert Selection)기능을 써보았을것이다. 이 선택항목은 마우스로 지적한 선택을 반전시키고 휴식상태로 강조한다. bash와 Korn은 또한 표현식으로 표현되는것들을 제외한 모든 파일이름들을 대신하는 류사한 기능을 제공한다. 실례로 표현식

```
!(*.exe)                .exe확장자가 없는 모든 파일들
```

은 .exe파일들을 제외한 모든것들을 정합한다. 만일 제외목록안에 다중표현식을 포함하고 싶다면 구분문자로 |를 사용하시오.

```
cp !(*.jpg|*.jpeg|*.gif) ../text
```

이것은 text등록부에 GIF또는 JPEG형식으로 된 도형파일들을 제외한 모든 파일들을 복사한다. 만일 묶음앞에 !가 있다면 괄호와 |는 파일이름들을 하나로 묶는데 리용될수 있다.

주해

배제기능은 shopt -s extglob를 설정하지 않는다면 bash에서 동작하지 않는다. 비록 이 의미를 리해하지 못하더라도 간단히 이 명령문을 시동파일들인(17.9.6) .bash_profile 또는 .profile 안에 넣어 보시오.

요 약

쉘은 사용자가 가입할 때 실행되는 지령이며 체계로부터 탈퇴할 때 완료된다. 쉘은 지령이 입력되기를 기다리며 특수한 문자들(메타문자들)이 있는가를 훑어 본다. 그리고 실행을 위하여 핵심부로 넘어 가기전에 지령행을 재조직한다.

쉘은 지령이 실행되기전에 확장되여야 하는 통용기호로 파일이름들을 정합한다. 그것은 여러개의 문자(*) 또는 한개의 문자(?)를 정합한다. 또한 범위([ ])를 정합할수 있으며 정합한것을 부정(!)할수도 있다. 이 문자들은 지령들에 무관계하다. 그러나 find는 자기의 통용기호를 가진다.

어떤 통용기호나 특수문자는 역사선(\)으로 의미를 해제하여(escape) 문자그대로 취급되도록 할수 있다. 인용부호로 의미해제를 진행하기도 한다. \은 또한 [Enter]건의 의미를 해제하여 긴 지령행을 여러개의 행들로 분할하도록 해준다.

때때로 역사선에 의한 의미해제(escaping)는 문자에 특수한 의미를 첨부(제거하는것이 아니라)하는 지령에 의해 사용된다. echo지령은 페지넘기기문자(\ f), 행바꾸기(\ n), 타브(\ t)를 출력하기 위한 특수한 확장문자렬(escape sequence)을 사용한다. echo는 또한 8진수값을 사용한다. echo \ 007은 삑소리를 낸다.

많은 지령들은 문자흐름의 형식으로 자료를 사용한다. 그것들은 표준입력흐름으로부터의 입력을 가지며 출력을 표준출력흐름에로 지적한다. 기정적으로 그것들은 각각 건반과 말단으로 설정된다. 그것들은 또한 디스크파일이나 관흐름으로부터 입력되거나 출력되도록 방향절환될수 있다.

기호 >는 존재하는 파일을 덧쓰기하며 >>는 표준출력을 방향절환함으로써 거기에 추가한다. <는 표

준입력의 방향을 절환한다. 표준입력과 표준출력을 사용하는 지령들을 려과기라고 부르는데 그것들은 UNIX체계안에 있다.

표준오유는 오유통보문들을 표현한다. 그의 기정목적지는 말단이지만 2>로 방향절환될수도 있는데 C 쉘에서는 아니다.

파일 /dev/null은 자료의 흐름이 그 파일에로 지적될 때조차 크기가 늘어 나지 않는 특수한 파일이다. /dev/tty는 모든 사용자들이 출력을 지적하기 위하여 사용할수 있는 모든 말단들에 대한 일반적인 장치이름이다.

관흐름을 리용하여 어떤 지령의 표준출력이 다른 지령의 표준입력에로 이어 질수 있다. 관흐름들에 놓인 려과기들의 결합은 지령들이 개별적으로 수행할수 없는 복잡한 과제들을 수행하는데 사용될수 있다.

tee지령은 2개의 흐름으로 출력을 가른다. 한 흐름은 표준출력으로 가며 다른것은 파일안에 복사된다. tee는 UNIX의 외부지령이지 쉘의 기능이 아니다.

지령대입은 어떤 지령의 출력이 다른 지령의 인수로 되도록 해준다. 그것은 한쌍의 역인용부호안에 지적된다.

쉘변수들은 스크립트론리(logic)에 리용되는 값들을 보관하는데 사용된다. 그것들의 형식은 변수=값인데 변수이름앞에 $을 붙임으로써 평가된다. UNIX체계의 동작을 조종하는 변수들을 환경변수라고 한다.

외인용부호는 모든 특수문자들을 보호하지만 겹인용부호는 변수평가와 지령대입을 가능하게 해준다.

쉘은 if, for, while과 같은 자체의 구조묶음을 가진 프로그람작성언어이기도 하다. 이 구조들은 UNIX 지령들과 쉘스크립트안에 있는 변수들과 결합된다. 쉘스크립트는 일반적으로 실행허가권을 요구한다.

C쉘(csh)이 의의 있는 기본기능들을 가지고 있다 하지만 Bourne쉘(sh)은 종합적인 쉘이다. Korn 쉘과 bash쉘은 Bourne쉘과 C쉘보다 우월한 측면도 가진다.

Korn쉘과 bash쉘은 Bourne쉘의 기능들을 정합하는 통용기호를 확장한다. 이 쉘들은 반점(,)을 패턴의 구분문자로 리용하면서 {}를 써서 여러 패턴들을 하나로 묶는다. !는 표현식에 맞는 파일들을 제외하고 나머지 모든 파일들을 선택하기 위하여 묶음연산자 ()와 구분문자 |를 함께 사용한다.

시험문제

1. 쉘은 왜 통용기호들을 확장시켜야 하는가?
2. 지령에 *가 하나의 인수로 있을 때 쉘은 무엇을 하는가?
3. 파일이름 chapa, chapb, chapc, chapx, chapy, chapz들을 한개의 표현으로 정합하시오.
4. rm *는 모든 파일들을 제거하는가?
5. 적어도 4개의 문자들을 가진 모든 파일이름들을 어떻게 렬거하는가?
6. 어느 UNIX지령이 자기 문법에 통용기호들을 사용하는가?
7. cat > foo를 사용할 때 만일 foo가 이미 어떤것들을 포함하고 있다면 무슨 일이 생기겠는가?
8. who >> foo를 사용할 때 foo가 존재하지 않는다면 무슨 일이 생기겠는가?
9. 여러개의 행들로 분할하고 싶은 긴 지령렬을 가지고 있다. 무엇을 주의해야 하는가?
10. 다음의 지령은 무엇을 의미하는가?

    >foo <bar bc

11. 오유통보문들이 말단에 나타나지 않게 하는 가장 좋은 방법은 무엇인가?
12. 지령프롬프트상에서 다음의 설정을 만드시오. $x를 실행시킬수 있는가?

x='ls | more' 혹은 set x='ls | more' (C쉘)

13. 지령 echo "$SHELL"과 echo '$SHELL'을 입력한다. 어떤 차이가 있는가?
14. 가입한 사용자들의 수를 어떻게 알아 내는가?
15. 변수값주기 x = 10(=의 량쪽에 공백)을 시도하시오. 만일 C쉘을 사용하지 않는다면 이 식이 타당한가?
16. directory='pwd'와 directory=`pwd`사이의 차이점은 무엇인가?
17. Linux에서 사용되는 표준쉘은 무엇인가?
18. 지령 echo "Enter your mame\c"가 Linux에서 프롬프트의 끝에 유표를 놓지 않았다. 왜 그런가?

련습문제

1. C쉘을 리용하지 않는다면 통용기호를 리용하여 첫번째 문자가 자모이며 마지막문자가 수자가 아닌 패턴을 형성하시오.
2. 지령 ls *.*의 의미는 무엇인가? 점을 포함하지 않는 파일도 정합하는가?
3. 패턴 .*.*[!.]를 생각하자. C쉘을 사용하지 않는다면 이 패턴을 정합하는 파일이름들안에 몇개의 점들이 있을수 있는가?
4. C쉘을 리용하지 않는 경우 등록부안의 숨겨 진 파일들만 제거하려면 어떻게 해야 하는가?
5. C쉘을 리용하지 않는다면 foo등록부안에 이음표로 시작하는 파일을 어떻게 제거하는가?
6. 표현 [3-h]*는 타당한가?
7. 점으로 시작하지 않는 파일들을 정합하시오.
8. ls .*swp는 파일이름 .ux.2.swp를 현시하는가?
9. 삑소리로 지령이 다 수행되였다는것을 알리려면 어떻게 해야 하는가?
10. cd *은 언제 작용하는가?
11. cat foo > foo를 리용할 때 무슨 일이 생기는가?
12. 지령 ls > newlist를 실행시킨다. newlist의 내용으로부터 무엇을 볼수 있는가?
13. 두개의 파일 foo1과 foo2를 련결하고 말단으로부터 그 사이에 어떤 본문을 삽입하려면 어떻게 해야 하는가?
14. wc < chap0[1-5]는 언제 작용하는가?
15. 지령 cat foo1 foo2 >/dev/tty의 출력은 표준출력에로 가는가?
16. wc를 두번 호출하여 생성된 다음의 두개 행사이에 차이점은 무엇인가? 왜 두번째 행에는 파일이름이 없는가?

```
3       20      103 infile
3       20      103
```

17. 려과기란 무엇인가? 려과기는 어디로부터 자기의 입력을 가지는가?
18. 겹인용부호리용의 두가지 결론은 무엇인가?
19. 지령대입을 리용하여 언제나 력서의 현재달을 인쇄하는 지령렬을 쓰시오
20. 지령대입을 하자면 그 지령이 려과기로 되여야 하는가?
21. 쉘스크립트 foo.sh는 who >/dev/tty를 포함한다. 그 지령의 출력이 말단으로 가는것과 관련하여 foo.sh > bar로써 스크립트를 방향절환할수 있는가?
22. 스크립트를 리용하지 않고 .bak확장자를 가지지 않는 모든 파일들을 foobar등록부안에 복사할수 있는가? 이 지령은 언제 동작하지 못하는가?

제 9 장. 려과기

이 장에서는 체계의 간단한 려과기들 즉 표준입력으로부터 자료를 받아 들이고 그 자료를 조작하여 표준출력을 만드는 지령들에 대하여 서술한다. 려과기는 UNIX도구묶음의 중심도구이며 이 장에서 소개되는 모든 려과기는 간단한 기능을 수행한다. 이 장은 려과기들이 독립적으로 리용되는 방식뿐아니라 방향절환과 관련결기능을 리용하여 다른 도구들과 결합되는 방법을 보여 준다.

많은 UNIX파일들은 마당 즉 어떤 의미 있는 실체를 표현하는 문자렬을 포함하는 행을 가지고 있다. 일부 지령들에서는 이 마당들이 자료로 리용되지 않는 적당한 구분문자에 의하여 분할되여야 한다. 대표적인 구분문자는 :(/etc/passwd와 $ PATH에서처럼)이며 이 장과 다른 장들에 있는 일부 실례파일들에 대해서는 구분문자로서 |(관련결)를 리용하였다. 대부분의 려과기들이 구분된 마당들을 가지고 동작하는데 일부는 그것들이 없이는 전혀 동작하지 못한다.

이 장에서 특색을 이루는 대부분의 지령들은 매우 중요하며 다른 장들에서도 광범히 리용된다. 이 지령들의 사용법을 리해한 다음에는 그것들이 단독으로는 수행할수 없는 과제 즉 내용조작과제를 수행하기 위하여 9.18을 읽는다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 한번에 한개 화면을 보는것과 more와 같이 pager로 패턴을 탐색한다(9.1).
- wc로 행, 단어,문자수를 계수한다(9.2).
- od로 8진수표현법을 적용하여 조종문자와 비인쇄문자들을 본다(9.3).
- pr로 여백과 머리부, 2줄공간, 다중렬출력을 제공하도록 본문을 양식화한다(9.4).
- cmp, diff, comm으로 두 파일의 차이점과 공통점을 찾는다(9.5부터 9.7까지).
- head로 시작행을, tail로 끝나는 행을 찾는다(9.8, 9.9).
- cut를 가지고 수직으로 문자나 마당을 자르고 paste로 두개 파일을 나란히 련결한다(9.10, 9.11).
- sort로 중복행들을 정렬하고 제거한다(9.12).
- tr로 개별적인 문자들을 변경, 삭제, 압축한다(9.13).
- uniq로 유일한 행과 유일하지 않은 행을 찾아 낸다(9.14).
- nl로 행에 번호를 준다(9.15).
- dos2unix와 unix2dos로 DOS와 UNIX파일사이의 변환을 수행한다(9.16).
- spell로 맞춤법검사를 진행하고 목록에 기입한다(9.17).
- 우의 모든 지령들을 결합하여 특별한 실례부분에서 내용조작과제를 수행한다(9.18).

Linux

- 보다 우월한 less의 페지화기능을 사용한다(9.1).
- 대화적으로는 물론 비대화적으로 맞춤법검사를 정확히 하는 ispell프로그람을 사용한다(9.17).

9.1 출력의 페지화(more)

man지령은 출력을 한번에 한페지씩 현시하는데 이것은 **페지화프로그람**(pager program)이 지원한다. UNIX는 두개의 페지화프로그람 more와 less를 제공하는데 초기의 페지화프로그람은 pg였다. more는 버클리에서 개발되였는데 오늘날에는 UNIX의 모든 판본에서 쓰이고 있다. less는 Linux에서 쓰이는 표준페지화프로그람(standard pager)이다. 우리는 이 절에서 more를 론의한다. Linux를 서술하는 절에서 less의 기능을 언급한다. 이 기능들의 대부분은 2.7에서 이미 론의되였다.

파일 chap01을 보려면 파일이름과 함께 more를 입력한다.

```
more chap01                완료하려면 q를 누르시오
```

그러면 chap01의 내용을 한번에 한페지씩 화면우에서 볼수 있다. 화면의 밑에 --More-- 라는 문자렬과 보기한 파일의 퍼센트가 보일것이다.

```
--More-- (17%)
```

이 시점에서 페지화프로그람을 완료하려면 q를 입력하시오. vi, emacs와 마찬가지로 more는 호출될때 화면에는 나타나지 않는 자체의 내부지령들도 가지고 있다. q는 내부지령이다.

항행(Navigation)

more의 항행기능은 체계에 따라 다르므로 자기의 체계에서 그것이 어떻게 적용되는가를 알려면 표 9-1과 2-1에 보여 준 지령들을 시험해 보아야 한다. 필요하다면 man페지들을 열람할수도 있고 more의 직결도움말기능(h를 리용)을 리용할수도 있다.

more는 판본에 관계없이 한번에 한페지 전진시키는 [spacebar]를 사용한다. 한페지를 앞으로 이동시키려면 f를 사용하고 한페지뒤로 이동하려면 b를 사용하시오.

f와 b를 포함하는 more에서 대부분의 지령들은 반복인자를 리용한다. 이것은 지령을 몇번 반복하도록 vi지령 4.6에 미리 설정된 수이다. 이것은 10페지 앞으로 이동하려면 10f를, 30페지뒤로 이동려면 30b를 리용할수 있다는것을 의미한다. 지령들 그자체는 화면에 어느 한 순간도 표시되지 않는다.

more는 자체의 반복지령도 가지고 있다. 이 지령은 마지막으로 리용한 지령을 반복하게 하는 점(vi에 리용된것과 같은 지령)이다. 만일 10f로 앞으로 이동한다면 간단히 점을 누름으로써 다른 방법으로 10페지를 이동할수 있다. 이것은 more에서 아주 편리한 기능이다.

패턴탐색

만일 vi를 잘 알고 있다면 새로운 패턴탐색기술을 배울 필요가 없다. 즉 more는 이 기술의 일부를 리용한다. 패턴을 찾는 방법을 요약하려면 간단히 /을 누르고 그다음 찾으려는 본문을 입력한다. 즉 UNIX라는 단어를 찾으려면 다음과 같이 리용할것이다.

```
/UNIX
```

만일 찾으려는 패턴이 없다면 바라는 행을 찾을 때까지 n을 누를수 있다. 일부 판본들은 역방향으로 탐색하기 위해 ?를 리용하도록 한다.

여기서 제공하는 패턴들은 간단한 문자렬로 제한하지 말아야 한다. 즉 그것들은 물론 정규식일수 있다. vi(4.15), emacs(5.13)와 함께 이 표현들을 리용하였다. 이 표현들은 행의 어느 위치에서든지 하나이상의 패턴을 정합하는 어떤 특별한 문자들을 사용한다. 이 두가지 탐색을 생각해 보자. 먼저 하나는 두

개의 문자렬을 정합하고 다음것은 쉘스크립트안에 있는 모든 지령행을 정합한다.

```
/[sS]ystem              system 혹은 System을 탐색한다
/^#             시작위치(^)에서 #를 탐색한다
```

다중파일이름을 사용하기

more는 또한 여러개의 파일이름과 함께 동작한다.

```
more chap01 chap02 chap03              more chap0[123]과 같다
```

먼저 첫번째 파일의 내용을 보여 준다. 이 파일의 보기가 끝난 다음 다음의 통보문을 보여 준다.

```
chap01: END (next file: chap02)
```

more는 이 프롬프트상에서 f를 누르거나 [spacebar]를 누르면 다음파일 chap02로 이동한다.
도중에 다음의 지령렬을 리용하여 다음 혹은 이전 파일로 절환할수 있다.

```
:n              다음파일
:p              이전 파일
```

more는 흔히 출력이 한개 화면안에 꽉 들어 차지 못하는 지령들과의 관흐름에 리용된다. 그것은 ls 지령과 함께 리용된다.

```
ls -l | more
```

more는 또한 매우 쓸모 있는 도움말기능을 가지고 있다. h를 누르면 그 화면이 호출되는데 모든 내부지령을 볼수 있을것이다. 중요한것은 표 9-1에 요약하였다.

참고

more는 자체로 vi편집기를 기동할수 있다. 간단히 v를 누른다. 편집을 끝낸 다음 more로 돌아 가려면 zz, :x 혹은 :wq를 사용하시오.

Linux

표준페지화프로그람 less

모든 Linux체계가 비록 more를 제공한다 해도 less는 그의 표준페지화프로그람이다. 사실 그것은 more의 상위모임이기때문에 같은 이름을 가진다는것은 우스운 일이다. 기능적인 측면에서는 vi에 가까운데 이것은 less를 배우는것이 vi사용자에 있어서는 배우기 쉽다는것을 의미한다.

항행은 vi와 호환성이 있다. 다음의 건들로 작업을 하여야 한다.

f, [Ctrl-f]혹은 [Spacebar]	한개 화면을 앞으로 흘러 보낸다
b 혹은 [Ctrl-b]	한개 화면을 뒤로 흘러 보낸다
j	한행 우로
k	한행 아래로

vi의 전통대로 less는 어떤 행으로 이동하기 위한 반복인자로서 G지령을 리용한다. 실례로 1G는 파일의 시작행으로 가게 되고 400G는 행번호 400으로 가며 G는 파일의 끝을 의미한다.

패턴탐색기술은 류사하다. more의 많은 판본들과 달리 less는 또한 ?패턴형식을 리용하여 반대방향에서 패턴을 탐색할수 있다. 그러나 less는 한가지 중요한 제한이 있다. more와 달리 마지막지령을 반복하지 않는다. 그러나 그의 가속기능은 100z에 의하여 100행씩 앞으로 뛰여 넘게 하며 오직 z(2.7-Tip)를 리용함으로써 지령을 반복한다.

표 9-1. more와 less의 내부지령(표 2-1도 볼것)

동 작	more	less
한페지 전진	[Spacebar] 혹은 f	[Spacebar] 혹은 f
20페지 전진	20f	-
한행 전진	[Enter]	j 혹은 [Enter]
1000행 전진	1000s	1000j
한페지 후진	b	b
15페지 후진	15b	-
1000행 후진	-	1000k
파일의 시작	-	p 혹은 1G
파일의 끝	-	G
300번째 행으로 가기	-	300G
pat에 대하여 정방향탐색	/pat	/pat
정방향탐색을 반복	n	n
pat에 대하여 역방향탐색	-	?pat
역방향탐색을 반복	-	N
지령행에 지적된 파일에로 뛰여넘기	:n	:n
지령행에 지적된 이전 파일에로 뛰여넘기	:p	:p
현재의 행번호를 현시	=	=
마지막지령을 반복	.(점)	-
vi편집기를 기동	v	v
UNIX지령 cmd를 실행	!cmd	!cmd
탈퇴	q	q
도움말보기	h	h

9.2 행과 단어, 문자의 계수(wc)

우리는 이미 wc지령을 많이 리용하였다. 이 지령은 사용된 선택항목에 따라 행, 단어, 문자들을 계수한다. 어떤 파일에 대해서 그것을 실행하자. 그러나 먼저 파일의 내용을 보자.

```
$ cat infile
I am the wc command
I count characters, words and lines
With options I can also make a selective count
```

wc는 파일에 대해 이 명령을 실행할 때 4개의 렬을 표시한다.

```
$ wc infile
        3     20     103 infile
```

지령은 3개 행, 20개 단어, 103개 문자를 계수하였다. 파일이름은 4번째 렬에 보여 주었다. 다음의 용어들의 의미는 이 책을 통하여 리용되므로 명백할것이다.

- **행**은 행바꾸기문자를 포함하지 않는 문자들의 모임이다.
- **단어**는 공백, 타브, 행바꾸기문자를 포함하지 않는 문자들의 모임이다.
- **문자**는 정보의 가장 작은 단위로서 모든 공백, 타브, 행바꾸기들을 포함한다.

wc가 지적된것만 계수하도록 할수 있는 다음의 3가지 선택항목들이 있다. -l선택항목은 행의 개수만

을 계수하고 -w, -c선택항목은 단어와 문자개수를 각각 계수한다.

```
$ wc -l infile
     3 infile             행의 수
$ wc -x infile
    20 infile             단어의 수
$ wc -c infile
  103 infile              문자의 수
```

여러개의 파일이름과 함께 사용할 때 wc는 매 파일별로 행을 만들고 총수를 맨밑에 현시한다.

```
$ wc chap01 chap02 chap03
  305    4058    23179  chap01
  550    4732    28132  chap02
  377    4500    25221  chap03
  1232  13290    76532  total           총수
```

응용

wc는 려과기이므로 어떤 UNIX지령의 표준출력에 있는 행, 단어들을 계수할수 있다. 실례로 어떤 등록부안에 있는 파일의 수나 사용자수를 셀수 있다.

```
ls | wc -1              파일의 수
who | wc -1             사용자의 수
```

여러개의 파일을 다룰 때 단어개수를 나타내는 하나의 수값에 흥미 있을수 있다. 이것은 이전에 리용한 원리(8.8)와 꼭같이 리용함으로써 쉽게 수행될수 있다. 즉 wc가 입력의 원천지를 무시하도록 만든다. cat로 여러 파일의 내용을 련결하고 출력을 wc -w에 흐름으로서 보낸다.

```
$ cat ux3rd?? | wc -w
303254
```

필요하다면 이것을 변수에도 설정할수 있다.

```
$ count=`cat ux3rd?? | wc w`
$ echo  $count
303254
```

그다음 이것을 쉘의 if명령문에서 조종지령으로 리용한다.

```
if [ $count -gt 30000 ] ; then
   echo "Your files have exceeded 30k"
fi
```

이 세 행들은 쉘프로그람의 범위에 속한다. 이런 구조에 UNIX지령과 관흐름을 리용할수 있다.

9.3 자료를 8진수로 현시하기(od)

대부분의 파일들(특히 실행파일)은 인쇄되지 않는 문자들을 포함하는데 UNIX지령들은 그것들을 합리적으로 현시하지 못한다. 아래에서 보다싶이 파일 odfile에는 일반적으로 보이지 않는 문자들이 들어있다.

```
$ more odfile
White space includes a
The ^G character rings a bell
The ^L cahracter skips a page
```

분명히 불완전한 첫행은 사실 [Tab]건을 눌러 입력된 타브문자를 포함한다. 이 문자를 볼수 있게 하려면 파일의 내용을 ASCII 8진수값(8진수체계)으로 보여 주는 od(octal dump)지령을 사용하여야 한다. -b선택항목은 매 문자들의 8진수값을 개별적으로 현시한다. 그 출력은 다음과 같다.

```
$ od -b odfile
0000000  127 150 151 164 145 040 163 160 141 143 145 040 151 156 143 154
0000020  165 144 145 163 040 141 040 011 012 124 150 145 040 007 040 143
.....
```

매행은 8진수로 된 16byte의 자료를 보여 주는데 그 파일의 위치는 행에서 첫 바이트에 놓인다. 그것들에 대하여 적당한 넘기기를 실시하지 않으면 이 출력을 가지고 뜻을 리해하기 어렵지만 -b선택항목과 -c(character)선택항목을 함께 쓰면 출력이 좀 더 리해하기 쉬워 진다.

```
$ od -bc odfile
0000000  127 150 151 164 145 040 163 160 141 143 145 040 151 156 143 154
           w   h   i   t   e       s   p   a   c   e       i   n   c   l
0000020  165 144 145 163 040 141 040 011 012 124 150 145 040 007 040 143
           u   d   e   s       a      \t  \n   T   h   e      \a       c
0000040  150 141 162 141 143 164 145 162 040 162 151 156 147 163 040 141
           h   a   r   a   c   t   e   r       r   i   n   g   s       a
0000060  040 142 145 154 154 012 124 150 145 040 014 040 143 150 141 162
           b   e   l   l  \n   T   h   e      \f       c   h   a   r
   .......
```

매행은 지금 두개 행으로 교체되였다. 8진수표현은 첫번째 행에 보여 주고 인쇄할수 있는 문자들과 확장문자렬(escape sequences)은 두번째 행에 그에 대응하게 보여 주었다. 파일의 첫번째 행에서 첫 문자는 W인데 이것은 8진수로 127이다. 그리고 다음의 문자들에도 주의를 돌려 보시오.

- 타브문자, [Ctrl-i]는 \t와 8진수 011로 나타난다.
- 삑소리문자, [Ctrl-g]는 \a와 8진수 007로 나타난다.
- 페지넘기기문자, [Ctrl-l]은 \f와 014로 나타난다.
- 행바꾸기문자, [Ctrl-j]는 \n과 012로 나타난다.

od는 행바꾸기문자도 보이게 한다. 이러한 확장문자렬들은 echo(8.5)와 함께 리용된다. 그것들은 C언어, awk, perl에 의해서 리용되기도 하므로 잘 알고 있어야 한다.

파일이름에서 비인쇄문자들을 검출하기

어디에 이 지령을 리용하는가? 때때로 파일이름이 인쇄할수 없는 문자를 포함하기때문에 파일을 호출할수 없는 경우가 있다.

```
$ ls p*
Program Files
```

여기에 공백은 있지만 어떻게 알수 있는가? 찾으려면 od를 리용하시오.

```
$ ls p* | od -bc
0000000  120 162 157 147 162 141 155 040 106 151 154 145 163 012
           P   r   o   g   r   a   m       F   i   l   e   s  \n
0000016
```

Program과 Files사이에 공백(8진수 040)이 있다. 이 파일을 제거하는것은 아주 쉬울것이다. 즉 **rm Program\ Files**로 하면 된다.

9.4 파일의 페지처리(pr)

pr지령은 파일에 머리부와 꼬리부형식화된 본문을 적당하게 추가함으로써 인쇄하기 위한 준비를 한다. 또한 많은 선택항목들을 가지고 있는데 일부는 아주 쓸모 있다. 먼저 선택항목들을 리용하지 않고 파일이름을 인수로 리용하여 이 지령을 해보시오.

```
$ pr group1


May 06 10:38 1999 group1              Page1


root:x:0:root                 이 7개 행들은 초기의 group1의 내용이다
bin:x:1:root,bin,daemon
users:x:200:henry,image,enquiry
adm:x:25:adm,daemon,listen
dialout:x:18:root,henry
lp:x:19:lp
ftp:x:50:
...빈 행들...
```

pr는 우아래에 5개 행의 여백을 추가한다. 페지의 아래부분은 지면상관계로 실례에서 보여 주지 않았다. 머리부는 파일이름, 페지번호, 파일의 마지막변경날자와 시간을 보여 준다.

pr의 출력은 화면의 내용을 미처 다 볼새없이 흐른다. 만일 이 절에서 실례로 실현해 보고 싶으면 역시 more나 less를 사용하는것이 더 좋다.

```
pr group1 | more
```

일반적으로 pr는 인쇄기로 자료를 내보내기전에 본문파일을 가공하기 위하여 리용된다. 만일 lp지령이 머리부를 인쇄하지 못한다면 《전처리기》(preprocessor)로서 pr를 결합하여 리용할수 있다.

```
$ pr group1 | lp
Request id is 334
```

pr의 선택항목

pr는 페지크기가 다른 경우에는 -l(length)선택항목으로 변화시킬수 있게 되여 있는 기정페지크기 66행을 가지고 있다. 또한 지적된 페지번호로부터 인쇄를 시작하도록 지시하게 되여 있다.

```
pr -l 72 chap01 | lp          한페지를 72행으로 설정
pr +10 chap01 | lp           10페지부터 인쇄한다
```

파일 group1을 종이에 두줄로 인쇄하는것이 어떤가? 그리고 파일로부터 모든 여백과 머리부를 완전히 삭제하는것이 어떤가? -t선택항목은 이러한 머리부들을 제거하고 -2는 두줄로 인쇄한다.

```
$ pr -t -2 group1
root:x:0:root                         dialout:x:18:root,henry
bin:x:1:root,bin,daemon               lp:x:19:lp
users:x:200:henry,image,enquiry       ftp:x:50:
adm:x:25:adm,daemon,listen
```

pr는 두줄공간본문(double-space text)으로도 출력할수 있다. 대체로 이 일감을 위해 sed지령을 리용하는데 -d(double)선택항목과 함께 pr지령을 쓰는것이 더 쉽다. 여백과 머리부를 삭제해 보자.

```
$ pr -t -d group1
root:x:0:root

bin:x:l:root,bin,daemon

users:x:200:henry,image,enquiry

adm:x:25:adm,daemon,listen
   .......
```

pr는 다른 선택항목들도 많이 가지고 있다. 행사이간격이 지정된 공간만큼 되도록 하기 위하여 -n으로 행간격을 두고 -o로 변위를 줄수 있다. pr의 중요한 선택항목들을 표 9-2에 보여 주었다.

참고

만일 인쇄출력이 너무 많은 공간을 차지하면 여러개의 렬로 인쇄하도록 pr의 -k선택항목을 리용하여 다시 형식화하는데 여기서 k는 옹근수이다

표 9-2. pr지령의 선택항목들

선택항목	의미
-l n	페지의 길이를 n개 행으로 설정한다
-w n	페지폭을 n개 문자로 설정한다
-h stg	매개 페지의 머리부를 문자렬 stg로 설정
-n	출력에 행번호를 붙인다
-on	n개 공백으로 출력의 편위를 지적한다
-d	두줄공간출력
-k	k개 렬로 출력을 생성한다
+k	k페지로부터 인쇄를 시작한다
-t	머리부, 꼬리부, 변두리여백을 완전히 삭제한다

9.5 두 파일의 비교(cmp)

흔히 두개의 파일이 같은가 알아 볼 필요가 생긴다. 그다음 둘중 하나는 지운다. UNIX체계에는 그러한 지령으로서 cmp, diff, comm지령이 있다. 이 절에서는 cmp(compare)지령을 볼것이다.

이 지령과 일부 다른 지령들의 리용을 실례로 보기 위하여 두개의 파일 group1과 group2를 리용하려는데 그것들사이에는 약간의 차이가 있다. 우리는 pr와 함께 group1을 사용했지만 두개의 파일을 나란히 볼 필요가 있다. 그것들을 그림 9-1에 보여 준다.

```
$ cat group1                      $ cat group2
root:x:0:root                     root:x:0:root
bin:x:1:root,bin,daemon           bin:x:1:root,bin,daemon
users:x:200:henry,image,enquiry   users:x:100:henry,image,enquiry
adm:x:25:adm,daemon,listen        adm:x:25:adm,daemon,listen
dialout:x:18:root,henry           dialout:x:19:root,henry
lp:x:19:lp                        lp:x:18:lp
ftp:x:50:                         ftp:x:50
                                  cron:x:16:cron
```

그림 9-1. 일부 차이점을 가진 두 파일 group1과 group2

두번째 파일은 추가적으로 한개 행을 더 가지고 있는데 인수로서 두개의 파일이름을 가지는 cmp를 리용하면 그 차이점을 알수 있다.

```
$ cmp group1 group2
group1 group2 differ: char 47, line 3
```

두 파일은 바이트별로 비교되는데 첫 같지 않는 위치(3번째 행의 47번째 문자)를 화면에 출력한다.

비록 3번째 행에서 불일치점이 먼저 발견되였지만 다른 곳에도 불일치점이 많다. -l(list)선택항목은 두 파일에서 서로 다른 매 문자들에 대해서 차이나는 8진수와 바이트수로 된 세부목록을 준다.

```
$ cmp -l group[12]                통용기호를 리용
  47  62  61
 109  70  71
```

```
   128  71  70
cmp: EOF on group1              group1이 먼저 끝났다
```

두 파일중 어느 한 파일이 끝에 도달할 때까지는 3개의 불일치점이 있다. 47번째 문자는 첫번째 파일에서 8진수로 62이고 다른 파일에서는 61이다.

만일 두 파일이 꼭 같다면 cmp는 통보문을 현시하지 않고 간단히 $프롬프트를 돌려 준다.

```
$ cmp group1 group1
$ _                 출력이 없다. 즉 파일들은 같다
```

두 파일사이의 차이점을 계수하도록 관흐름(pipeline)을 설정하여 cmp를 리용할수 있다.

```
$ cmp -1 group? | wc -1
3
```

mp는 차이가 존재하는 수자적인 위치를 알려 주는데 이것은 두 파일이 같은가 다른가를 알기 위한 가장 좋은 방법이라는것외에 더이상 도움을 주는것이 없다. 차이점을 포함한 행들을 식별하기 위하여 diff지령이 필요하다.

9.6 한 파일을 다른 파일로 변환하기(diff)

diff는 파일들사이의 차이를 현시하는 다른 방법을 가지고 있다. cmp와는 달리 이 지령은 두파일을 같아 지도록 하자면 한 파일의 어느 행들을 변화시켜야 하는것도 알려 준다. 같은 파일들을 사용하면 상세한 출력을 내보낸다.

```
$ diff group[12]
3c3                                   첫번째 파일의 3번째 행을 변화시키는데 구체적으로는
< users:x:200:henry,image,enquiry     이 행을 아래의
--
> users:x:100:henry,image,enquiry     이 행으로 교체한다
5,6c5,6                               5행부터 6행을 변화시키는데
< dialout:x:18:root,henry             이 두개 행을 아래의
< lp:x:19:lp
---
> dialout:x:19:root,henry             이 두개 행으로 교체한다
> lp:x:18:lp
7a8                                   첫번째 파일의 7행다음에 아래의
> cron:x:16:cron                      이 행을 추가한다
```

diff는 두 파일이 같아 지도록 변화시켜야 한다는것을 지적하기 위하여 **명령**(instruction)들과 함께 어떤 특수한 기호들을 사용한다. 이 명령들은 체계에서 가장 위력한 지령(command)들중의 하나인 sed 지령에 리용된것과 류사한것으로 리해할수 있다.

매 명령들은 첫번째 파일에 적용되며 주소(address)와 동작(action)으로 구성된다. 명령 3c3은 3행을 어느 한 행으로 변화시키는데 변환후에는 3행이 남아 있다. 7a8은 7행다음에 한개 행을 추가하는데

두번째 파일에는 행번호로서 8을 추가한다. 5,6c는 2개 행을 변화시킨다.

참고

만일 두 파일이 같은가를 간단히 판단하려면 아무러한 선택항목도 없이 cmp를 사용하시오.

9.7 공통적인 부분을 찾기(comm)

사람이름이 들어 있는 두 목록을 가지고 있다고 가정하고 한 목록에는 있지만 다른 목록에는 없는 이름을 찾아 보자. 두 목록에 다 있는것도 찾아 보자. comm은 이러한 작업에 필요한 지령이다. 이것은 두개의 정렬되여 있는 파일들을 요구하는데 차이나는 항목들을 서로 다른 렬들에 렬거한다.

```
$ cat foo1                         $ cat foo2
charlie                            bob
henry                              charlie
julie                              harry
monty                              julie
sumit                              monty
                                   sumit
```

두 파일은 정렬되여 있으며 일부 차이점들을 가지고 있다. comm을 실행시키면 3개의 렬형태로 출력을 현시한다.

```
$ comm foo1 foo2
     bob                 두번째 파일에만 있는것
          charlie                두 파일에 공통적인것
     harry
     henry                첫번째 파일에만 있는것
          julie
          monty
          sumit
```

첫번째 렬은 첫번째 파일에만 있는 한개 행을 포함하며 두번째 렬은 두번째 파일에만 있는 두개 행을 포함한다. 세번째 렬은 두 파일에 공통적인 4개 행들을 현시한다.

이 출력은 크게 쓸모 없지만 comm은 선택항목 -1, -2, -3을 리용하여 선택적인 출력을 생성할수도 있다. 필요한 렬만 현시하려면 간단히 -부호와 함께 렬번호를 리용한다. 또한 선택항목들을 결합할수도 있으며 공통인 행들만 현시할수도 있다.

```
comm -3 foo1 foo2              두 파일들에 공통적이지 않는 행들을 선택한다
comm -13 foo1 foo2              두번째 파일에만 있는 행들을 선택한다
```

마지막실례와 그외에 다른 선택항목(-23)은 생각한것보다 더 특별한 값을 가지고 있다. 우리는 이 장의 끝부분에서 이 지령을 리용한 실례를 고찰할것이다.

9.8 파일시작부분을 현시하기(head)

head명령은 그 이름이 보여 주는바와 같이 파일의 맨 웃부분을 현시한다. 선택항목이 없이 사용하면 인수파일의 첫 10개 행을 현시한다.

```
head foo                 foo의 첫 10개 행을 보여 준다
```

행수를 지정하여 실례로 첫 3개 행을 현시할수도 있다. -기호를 리용하는데 뒤에 수값인수가 따른다.

```
$ head -3 group1
root:x:0:root
bin:x:1:root,vin,daemon
users:x:200:henry,image,enquiry
```

head는 상상적인 방법으로 리용될수 있다. 실례로 다음날 편집을 계속하려고 하는데 마지막으로 편집된 파일을 다시 호출할수 없을수도 있다.

ls -t는 변경시간순서로 파일들을 현시하여 목록으로부터 첫번째 파일을 꺼내고 그것을 vi편집기의 인수로 리용하여 그 일감을 수행할것이다. 이것은 지령대입을 요구한다.

```
vi `ls -t | head -1`          편집을 위해 마지막으로 변경된 파일을 연다
```

사용자는 이것을 별명(17.4)으로 정의하여 그 별명지령을 사용할수 있다.

head는 몇개의 행들로 현시되도록 제한하기 위하여 흔히 grep지령(15.2)과 함께 리용된다. 다음의 지령렬은 단어 IMG SRC다음에 문자렬 GIF를 포함하고 있는 첫 5개 행을 꺼낸다.

```
grep "IMG SRC.*GIF" quote.html | head -5
```

여기서 우리는 어떤 문자들의 리용을 지적하기 위하여 정규식 .*를 리용하였다. 이 기호는 SRC와 GIF사이에 어떤 문자들이 있을수 있다는것을 암시한다(전혀 없을수도 있다).

Linux

GNU head는 파일의 시작위치로부터 일정한 수의 문자, 블로크, KB(키로바이트)와 MB(메가바이트)를 꺼낼수 있다. 그래서 만일 cmp가 47번째 문자가 어느 곳에 위치하고 있는가를 말해 주지 못한다면 head의 -c(character)선택항목은 차이나는 위치를 정확히 보여줄것이다.

```
$ head -c47 group1
root:x:0:root
bin:x;l:root,bin,daemon
users:x:2
```

이것들은 group1의 47개 문자이다. 만일 여기서 마지막문자를 group2와 비교하면 거기에 1(그룹ID는 거기서 100이다.)이 있다는것을 알수 있다. 다른 부분에서도 꺼낼수 있다.

```
head -c 1b shortlist          첫 512byte블로크
head -c 2m README             2MB
```

cmp와 같은 일부 프로그람들은 파일안에 있는 어떤 문자의 위치에서 오유와 차이점을 지적하는데 여기에 -c선택항목이 매우 쓸모 있다.

9.9 파일끝부분을 현시하기(tail)

head와 반대로 tail은 파일의 끝을 현시한다. 이것은 head와 같이 행을 주소화하는 추가적인 방식을 제공하는데 인수없이 리용될 때에는 마지막 10개 행을 현시한다. 이런 식으로 마지막 3개 행을 현시할수도 있다.

```
$ tail -3 group1
dialout:x:18:root,henry
lp:x:19:lp
ftp:x:50:
```

UNIX의 일부 판본들에서는 tail이 꺼낼수 있는 토막의 크기가 제한되여 있다. 이 문제를 극복하기 위하여 파일의 끝대신에 파일의 시작위치로부터 행을 주소화할수 있다. +k선택항목은 그렇게 하도록 허락하는데 여기서 k는 선택하기 시작한 곳으로부터 행수를 표현한다. 만일 파일이 1000행을 포함하고 있다면 마지막 200개 행을 선택하는것은

```
tail +801 foo                801번째 행으로부터 우로, +기호로 가능
```

을 리용한다는것을 의미한다.

tail은 블로크나 문자단위로도 꺼낼수 있다. 다음의 지령은 마지막 512byte를 꺼낸다.

```
tail -512c foo               블로크에 대해서는 b를 리용한다
```

tail지령의 판본들은 대체로 -r선택항목을 리용하여 현재행들을 반대순서로 꺼내기도 한다.

파일증가를 조절하기(-f)

대부분의 UNIX프로그람들은 그것들이 실행되고 있는 동안 체계의 기록파일(log file)들에 계속 쓰기한다. 체계관리자는 맨 마지막통보문을 보려면 이 파일들의 증가량을 조절해야 할 필요가 있다. tail은 이러한 목적을 위해 -f(follow)선택항목을 제공한다. 다음의 례는 다른 말단으로부터 기록파일 install.log의 증가량을 살펴 봄으로써 Oracle 8.1의 설치를 감시할수 있는 방법이다.

```
tail -f /oracle/app/oracle/product/8.1/orainst/install.log
```

프롬프트는 작업이 끝난 다음에도 귀환되지 못한다. 쉘에로 탈퇴하기 위하여 이 선택항목으로 그 프로쎄스를 중단하여야 한다. 자기 기계에 적용할수 있는 새치기건을 리용하시오.

참고

파일에 련속적으로 쓰는 프로그람을 실행하고 있을 때 파일이 얼마만큼 증가하는지 보고 싶으면 tail -f를 사용하시오. 새치기건으로 이 지령을 완료하여야 한다.

Linux

행이 아니라 바이트를 꺼내기

GNU tail도 head에 대한 절에서의 Linux해설부에서 론의된 선택항목들을 공유한다.

9.10 파일을 수직으로 가르기(cut)

head와 tail이 파일을 수평으로 자르는데 리용된다면 cut지령으로는 파일을 수직으로 자를수 있다. 먼저 렬을 보자.

렬자르기(-c)

파일묶음의 첫 4개 렬을 추출하기 위해 cut를 사용하자. -c선택항목뒤에 렬을 지적하는것이 필요하다.

```
$ cut -c1-4 group1              -c나 -f선택항목은 항상 필요하다
root
bin:
user
adm
dial
lp:x
ftp:
```

-c1-4는 1렬부터 4렬까지 자른다. 하나이상의 렬을 지적하면서 cut를 리용할수도 있다. 범위가 허용되면 반점들은 다수의 렬들로 가르는데 리용될수 있다.

```
cut -c -3,6-22,28-34,55- foo              증가하는 렬이여야 한다
```

렬목록에는 공백들이 없어야 한다. cut는 파일의 시작위치부터 끝위치로 가면서 렬을 선택하기 위하여 특별한 형식을 사용하기도 한다는것을 주목하시오. 표현 55-는 렬번호 55부터 행의 끝렬까지 지적한다. 마찬가지로 -3은 1-3과 같다.

마당자르기(-f)

-c선택항목은 고정길이로 된 행들에 쓸모 있다. 대부분의 UNIX파일들(/etc/passwd와 /etc/group와 같은 파일)은 고정길이의 행들을 포함하지 않는다. 이 경우에 렬보다도 마당들을 자르는것이 필요하다.

cut는 타브를 기정마당구분문자로 리용하는데 다른 구분문자들로 작업할수도 있다. 여기서는 2개의 선택항목 즉 구분문자를 위해 -d, 마당목록지정을 위해 -f선택항목이 필요하다. 아래의것은 첫번째와 세번째 마당을 자르는 방법을 보여 준다.

```
$ cut -d: -f1,3 group1
root:0
bin:1
users:200
adm:25
dialout:18
lp:19
ftp:50
```

-f선택항목을 리용할 때 파일이 기정구분문자(tab)를 가지고 있지 않다면 -d선택항목을 사용해야 한다.

cut는 구분문자로서 공백을 지정함으로써 행의 첫 단어를 뽑아 내는데 리용될수 있다. 3.2에서 리용된 첫번째 실례는 지금 cut와 함께 실행되여 오직 사용자들의 목록만을 현시한다.

```
$ who | cut -d " " -f1          공백은 구분문자이다
root
romeo
andrew
juliet
```

cut는 다른 지령이나 려파기와의 결합에서 자주 리용되는 위력한 본문조작자이다.

주해 cut에 마당을 추출하겠는가 렬을 추출하겠는가를 지적하여야 한다. -f와 -c선택항목중 어느 하나는 지정되여야 한다. 이 선택항목들은 선택적이지 않다. 즉 그것들중 하나는 의무적이다.

9.11 파일의 붙이기(paste)

cut로서 자른것은 반대로 paste지령으로 수평으로가 아니라 수직으로 붙일수 있다. 그것들을 붙임으로써 2개의 파일을 나란히 볼수 있다. cut와 같이 paste도 구분문자를 지적하기 위해 -d선택항목을 사용하는데 기정적으로는 타브이기도 하다.

8.6에서 만든 두 파일 calc.lst 와 result.lst 를 문자그대로 련결하기 위해 paste를 사용하시오. 구분문자로 =를 리용한다.

```
$ paste -d= calc.lst result.lst
      2 ^ 32=4294967296
     25 * 50=1250
30*25 + 15^2=975
```

비록 paste가 행을 련결하기 위하여 적어도 두개의 파일을 사용한다 해도 그자체가 려파기이기도 하다는것을 잊지 말아야 한다. 이것은 어떤 파일에 대한 자료가 표준입력을 통해 들어 올수 있다는것을 의미한다. bc가 표준출력에 그의 출력을 내보내므로 bc와 paste를 관흐름에서 쓸수 있다.

```
$ bc < calc.lst | paste -d= calc.lst -
      2 ^ 32=4294967296
     25 * 50=1250
30*25 + 15^2=975
```

보통 이것이 더 좋다. 즉 이때에는 어떠한 중간파일도 만들지 않는다. -기호는 paste의 두번째 인수가 표준입력으로부터 들어 와야 한다는것을 지적한다. calc.lst(paste와 함께 리용되는것)와 -의 위치를 바꿀수도 있다.

9.12 파일정렬(sort)

UNIX의 sort는 보통 정렬기능을 수행하며 변수길이행에 잘 쓰인다. 여러개의 선택항목을 가지고 있는데 모든 가능한 방법중에서 shortlist파일을 정렬함으로써 중요한것들만 고찰하겠다. 이 파일은 개인자료에 대한 5개 행을 포함한 본문파일이다.

```
$ cat shortlist
2233|charles harris  |g.m.      |sales      |12/12/52| 90000
9876|bill johnson    |director  |production |03/12/50|130000
5678|robert dylan    |d.g.m     |marketing  |04/19/43| 85000
2365|john woodcock   |director  |personnel  |05/11/47|120000
5423|barry wood      |chairman  |admin      |08/30/56|160000
```

매행은 |에 의하여 6개 마당으로 구분되여 있다. 종업원의 세부정보는 매행에 보관되여 있다. 매 성원은 종업원ID, 이름, 직위, 부서, 생년월일, 로임으로 식별되여 있다. 이 파일은 읽기 쉽게 고정된 형식으로 신중히 설계되였다(UNIX도구묶음으로 가능한 조작범위를 보기 위하여 제15장에서는 이 파일의 확장된 판본을 리용한다).

sort가 선택항목없이 호출되면 전체 행이 정렬된다.

```
$ sort shortlist
2233|charles harris  |g.m.      |sales       |12/12/52| 90000
2365|john woodcock   |director  |personnel   |05/11/47|120000
5423|barry wood      |chairman  |admin       |08/30/56|160000
5678|robert dylan    |d.g.m     |marketing   |04/19/43| 85000
9876|bill johnson    |director  |production  |03/12/50|130000
```

정렬은 매행의 첫번째 문자부터 시작하는데 두 행에서 문자들이 같을 때에만 다음문자로 옮긴다. 기정적으로 sort는 행의 시작위치에서 시작하여 ASCII순서맞추기렬로 행을 재정렬한다. 이 기정정렬순서는 어떤 선택항목을 리용하여 바꿀수 있다.

sort의 선택항목

마당에 기초한 정렬(-t)

cut, paste와 같이 sort도 마당우에서 작업하는데 기정마당구분문자는 공백(cut와 paste에서는 타브)이다. -t선택항목을 리용하여 파일을 어떤 마당 례하면 두번째 마당(이름)에 한해서 정렬할수 있다

```
$ sort -t \| +1 shortlist                -t "|"도 사용할수 있다
5423|barry wood      |chairman  |admin       |08/30/56|160000
9876|bill johnson    |director  |production  |03/12/50|130000
2233|charles harris  |g.m.      |sales       |12/12/52| 90000
2365|john woodcock   |director  |personnel   |05/11/47|120000
5678|robert dylan    |d.g.m     |marketing   |04/19/43| 85000
```

|는 쉘이 그것을 관흐름문자로 해석하는것을 막기 위하여 \으로 의미해제되여야 한다. 인수 +1은 정렬이 첫번째 마당을 건너 뛰고 정렬을 시작하라는것을 지적한다. 세번째 마당을 분류하기 위해 다음의 지령을 리용한다.

```
sort -t "1" +2 shortlist
```

정렬순서는 -r(reverse)선택항목으로 반전될수 있다. 다음의 지령렬은 이전의 정렬순서를 반전시킨다.

```
$ sort -t "|" -r +1 shortlist
5678|robert dylan    |d.g.m     |marketing   |04/19/43| 85000
2365|john woodcock   |director  |personnel   |05/11/47|120000
2233|charles harris  |g.m.      |sales       |12/12/52| 90000
9876|bill johnson    |director  |production  |03/12/50|130000
5423|barry wood      |chairman  |admin       |08/30/56|160000
```

sort는 특별한 방식으로 선택항목들을 결합한다. 이전의 지령렬은 다음과 같이 씌여질수도 있을것이다.

```
sort -t "|" +1r shortlist                두번째 마당을 반대로 배렬
```

주해

기정적으로 정렬은 다음의 순서 즉 수자, 대문자, 소문자순서로 진행된다. 어떤 선택항목을 sort와 함께 리용함으로써 이 순서를 변경시킬수 있다.

보조열쇠에 기초한 정렬

한개이상의 마당에 대해서도 정렬할수 있다. 즉 sort에 보조열쇠를 제공할수 있다. 만일 기본열쇠(primary key)가 세번째 마당이고 보조열쇠가 두번째 마당이면 다음의 지령렬을 리용할수 있다.

```
$ sort -t \| +2 -3 +1 shortlist
5423|barry woo       |chairman  |admin       |08/30/56|160000
5678|robert dylan    |d.g.m     |marketing   |04/19/43| 85000
9876|bill johnson    |director  |production  |03/12/50|130000
2365|john woodcock   |director  |personnel   |05/11/47|120000
2233|charles harris  |g.m.      |sales       |12/12/52| 90000
```

이것은 직위와 이름으로 파일을 정렬한다. -3은 세번째 마당다음에 정렬을 중지하라는것을 지적하고 +1은 첫번째 마당다음에 정렬을 계속하라는것을 지적한다. 첫번째 마당으로부터 정렬을 다시 시작하려면 +0을 사용한다.

주해

+1은 정렬을 첫번째 마당으로부터가 아니라 두번째 마당으로부터 시작하라는것을 의미하며 -4는 네번째 마당다음에는 정렬을 완료하라는것을 의미한다.

렬에 기초한 정렬

정렬의 시작위치를 지적하기 위하여 마당안에 있는 문자위치를 지적할수 있다. 만일 생년월일에 따라 파일을 분류하여야 한다면 다섯번째 마당안에서 7번째와 8번째 렬에 관하여 정렬하는것이 필요하다.

```
$ sort -t"|" +4.6 -4.8 shortlist
```

```
5678|robert dylan   |d.g.m     |marketing   |04/19/43| 85000
2365|john woodcock  |director  |personnel   |05/11/47|120000
9876|bill johnson   |director  |production  |03/12/50|130000
2233|charles harris |g.m.      |sales       |12/12/52| 90000
5423|barry wood     |chairman  |admin       |08/30/56|160000
```

여기서 렬명세의 해석은 아주 특이하다. +4.6은 정렬시작위치 즉 다섯번째 마당의 7번째 렬을 의미한다. 이처럼 -4.8은 정렬이 같은 마당의 8번째 렬다음에서 멎어 선다는것을 의미한다.

수값에 기초한 정렬(-n)

sort가 수자에 립각하여 동작할 때 이상한 현상이 일어 날수 있다. 세번째 마당(수값그룹ID를 포함)에서 묶음파일을 정렬할 때 이상한 결과가 얻어 진다.

```
$ sort -t: +2 -3 group1
root:x:0:root
bin:x:1:root,bin,daemon
dialout:x:18:root,henry
lp:x:19:lp
users:x:200:henry,image,enquiry                25우에 200
adm:x:25:adm,daemon,listen
ftp:x:50:
```

이것은 아마 기대한것이 아니지만 ASCII순서맞추기렬은 200을 25우에 배치한다(0은 5보다 더 작은 ASCII값을 가지고 있다). 이것은 -n(numeric)선택항목에 의해 무시될수 있다.

```
$ sort -t: +2 -3 -n group1           +2n -3을 사용할수도 있다
root:x:0:root
bin:x;1:root,bin,daemon
dialout:x:18:root,henry
lp:x:19:lp
adm:x:25:adm,daemon,listen
ftp:x:50:
users:x:200:henry,image,enquiry
```

참고 수값마당에 관하여 파일을 정렬할 때에는 언제나 -n선택항목을 사용하시오. 만일 명백한 ASCII정렬을 요구하는 다른 정렬마당이 있다면 수값정렬을 요구하는 렬명세서에 +2n과 같이 n을 뒤붙이하시오.

중복행제거(-u)

-u(unique)선택항목은 파일에서 중복행을 제거한다. shortlist에서 직위마당을 잘라 버린다면 파일안에서 생긴 유일한 직위들을 찾아 내기 위하여 sort에 그것을 관련결할수 있다.

```
$ cut -d"|" -f3 shortlist | sort -u | tee desigx.lst
chairman
d.g.m.
```

```
director
g.m.
```

우리는 본문조작문제를 해결하기 위하여 세개의 지령들을 사용하였다. 여기서 cut는 정렬된 출력으로부터 세번째 마당을 선택하는데 쓰인다.

sort의 다른 선택항목들

sort는 려과기이기도 하므로 정렬된 출력은 >연산자와 함께 어떤 파일로 방향절환(redirection)될수 있다. sort와 uniq는 출력파일이름을 인수로 받아 들이기도 하는 유일한 UNIX려과기들이다. sort는 이 과제를 위하여 -o(output)선택항목을 리용하는데 이상하게도 입력과 출력파일이름이 꼭 같아 질수 있다.

```
sort -o sortedlist +3 shortlist            sortedlist안에 보관되는 출력
sort -o shortlist shortlist             같은 파일안에 보관되는 출력
```

그리고 파일이 실지 정렬되였는가를 검사하고 싶으면 -c(check)선택항목을 쓸수 있다.

```
$ sort -c shortlist
$ _                 파일이 보관된다
```

가령 지금 파일이 네번째 마당에 관하여 정렬되였는가를 검사하고 싶다고 하자. sort는 그것을 명백히 지적한다.

```
$ sort -t \| +3 -c shortlist
sort: shortlist:2: disorder: 9876|bill johnson  |director |production |03/12/50
|130000
```

지령행에 한개이상의 파일이 있으면 sort는 우선 파일들을 련결하고 그다음 그것들을 집체적으로 정렬한다. 큰 파일들을 이런 방법으로 정렬할 때 성능이 흔히 떨어 진다. 때때로 sort의 -m(merge)선택항목의 기능을 리용하여 그것들을 통합하기전에 개별적으로 파일들을 정렬하는것이 더 낫다. 다음의 지령은 개별적으로 정렬된 3개의 파일을 합친다.

```
sort -m foo1 foo2 foo3
```

중요한 sort선택항목은 표 9-3에 요약되여 있다.

표 9-3. sort의 선택항목들

선택항목	의미
-tchar	마당을 구별하기 위하여 구분문자 char를 리용한다
-o flname	flname안에 출력을 배치한다
+k	k번째 마당을 뛰여 넘은 다음 정렬을 시작한다
-k	k번째 마당다음에 정렬을 중지한다
+m.n	m+1번째 마당의 n번째 렬다음부터 정렬을 시작한다
-m.n	m+1번째 마당의 n번째 렬에서 정렬을 중지한다
-f	소문자를 그에 대응한 대문자로 본다(대소문자구별이 없는 정렬)
-r	정렬순서를 반전한다
-c	파일이 정렬되여 있는가를 검사한다
-n	수값으로 정렬한다
-m list	list안에 있는 정렬된 파일들을 통합한다
-u	중복행을 제거한다

9.13 문자번역(tr)

지금까지의 지령들은 전반적인 행이나 렬을 다루고 있다. tr(translate)지령은 문자흐름에서 개별적인 문자들을 조작한다. 이 지령은 하나 혹은 두개의 간결한 표현을 사용하여 문자들을 번역하며 특별한 문법을 가진다.

```
tr 선택항목 표현1 표현2 < 표준입력
```

이 지령은 오직 표준입력으로부터만 입력을 가진다는것을 주의하시오. 파일이름은 인수로 가지지 못한다. 기정적으로는 표현1안에 있는 모든 문자를 표현2안에 있는 대응되는 문자들로 번역한다.

첫번째 표현에서 첫 문자는 두번째 표현의 첫번째 문자로 교체되며 다른 문자들도 같다.

|를 ~(tilde)로, /을 -로 교체하기 위하여 tr를 사용할수 있다. 간단히 해당한 렬안에 그것들을 담고 있는 두 표현을 지적한다.

```
$ tr '|/' '~-' < shortlist | head -3
2233~charles harris  ~g.m.      ~sales      ~12/12/52~ 90000
9876~bill johnson    ~director  ~production ~03/12/50~130000
5678~robert dylan    ~d.g.m     ~marketing  ~04/19/43~ 85000
```

두 표현의 길이가 같다는데 주의를 돌리시오. 만일 길이가 다르다면 보다 긴 표현은 대응되지 않는 문자들을 가질것이다(Linux에서는 아니다). 여기서 외인용부호는 변수평가나 지령대입이 포함되지 않기때문에 리용되였다. 두 표현을 두개의 변수로서 정의할수 있으며 그때 겹인용부호로 그것들의 값을 구한다.

```
exp1='|/' ; exp2='~-'
tr "$exp1" "$exp2" < shortlist                  오직 겹인용부호로만 변수값구하기
```

통용기호와 같이 tr도 그것을 사용하는 표현안에서 범위를 받아 들인다.같은 규칙들이 적용된다. 즉 -(이음표)의 오른쪽에 있는 문자는 왼쪽에 있는것보다 더 큰 ASCII값을 가져야 한다. \ 에 의한 의미해제규칙들도 명백할것이다. [문자는 그의 특수한 의미가 제거되여야 하는 경우에는 \ 에 의하여 의미를 해제해야 한다.

tr가 파일이름을 인수로 받아들이지 못하므로 입력은 파일에로 방향절환되여야 한다. 그리고 관흐름을 통하여 제공될수도 있다. 아래에서 첫 세개 행의 소문자를 대문자로 변화시키는데 이 지령을 리용한다.

```
$ head -3 shortlist | tr '[a-z]' '[A-Z]'
2233|CHARLES HARRIS  |G.M.     |SALES       |12/12/52| 90000
9876|BILL JOHNSON    |DIRECTOR |PRODUCTION  |03/12/50|130000
5678|ROBERT DYLAN    |D.G.M    |MARKETING   |04/19/43| 85000
```

두 표현식을 반전시키면 대문자를 소문자로 변환할것이다. tr는 흔히 파일안의 문자들을 변화시키는데 리용된다.

주해

tr는 파일이름을 인수로 받아 들이지 못하지만 방향절환(redirection)이나 관흐름을 통하여 입력을 가진다(표준입력만).

tr의 선택항목

문자지우기(-d)

shortlist파일은 구분문자로 분리되는 마당과 /을 가지고 읽기 가능한 형식으로 양식화된 날자마당을

가지고 있다. 자료기지가 아닌 설치판들에서는 결코 구분문자들이 리용되지 않으며 날자표시는 여전히 mmddyy형식으로서 6개 문자로 된 마당이 리용된다. 만일 이 파일을 전통적인 형식으로 변화시키는것이 필요하다면 파일로부터 |와 /문자들을 지우기 위하여 -d(delete)선택항목을 리용한다. 다음의 지령은 첫 세개 행에 대해서 그것을 수행한다.

```
$ tr -d '|/' < shortlist | head -3
2233charles harris  g.m.     sales      12/12/52 90000
9876bill johnson    director production 03/12/50130000
5678robert dylan    d.g.m    marketing  04/19/43 85000
```

반복적인 문자들을 압축하기(-s)

UNIX도구들은 흔히 마당들로 작업한다 그러므로 오히려 분리된 마당들을 가진 파일을 리용하는것이 더 좋을것이다. 또한 모든 여가공백들을 제거함으로써 총적으로 압축시킬수 있다. -s(squeeze)선택항목은 인수의 반복적인 발생을 한개 문자로 압축시킨다.

```
$ tr -s ' ' <shortlist | head -3                공백들을 압축시킨다
2233|charles harris |g.m. |sales |12/12/52| 90000
9876|bill johnson |directory |production|03/12/50|130000
5678|robert dylan |d.g.m. |marketing |04/19/43| 85000
```

표현을 보충하기(-c)

마지막으로 -c(complement)선택항목은 표현에서 문자들의 묶음을 보충(부정)한다. |과 /을 제외한 모든 문자들을 지우려면 -d와 -c선택항목을 결합할수 있다.

```
$ tr -cd '|/' < shortlist
||||//|||||//|||||//|||||//|||||//|||||//|||||//|||||//|||||//|||||/
/|||||//|||||//|||||//|||||//|||||//|$ _
```

출력이 매우 특이하다. tr는 파일에서 |와 /을 제외한 모든 문자들을 지웠다. 행바꾸기문자도 없어졌다. 이것은 다음행이 아니라 출력끝에 나타나는 프롬프트에 의하여 지적되여 있다. 우리는 실례부분(9.18)에서 개별적인 행안에 매 단어들을 배치하기 위하여 이 -cd선택항목결합을 리용할것이다.

8진수값을 사용하기

echo와 같이 tr도 문자의 ASCII8진수값을 사용한다. 그러므로 매행들에 매 마당을 현시하고 싶다면 |를 행바꾸기문자(8진수 012)로 교체할수 있다.

```
$ tr '|' '\012' < shortlist | head -6
2233
charles harris
g.m.
sales
12/12/52
90000
```

Linux

tr에서 확장문자렬을 리용하기

앞의 지령은 Linux에서 다음과 같이 GNU의 tr로 씌여 질수 있다.

```
tr '|' '\n' < shortlist | head -6
```

Linux도 echo지령(8.5)이 리용하는 일반적인 확장문자렬(escape sequences)을 모두 리용한다.

9.14 반복 및 비반복행들의 위치알아내기(uniq)

파일들을 련결하거나 통합할 때 안에 들어 있는 항목들이 중복되는 문제들에 주의를 돌려 보았을것이다. sort가 -u선택항목으로 그것들을 제거하는 방법들을 보았다. UNIX는 이런 행들을 조종하기 위한 특별한 도구 즉 uniq지령을 제공한다. 이 지령은 관흐름안에 놓일 때 가장 쓸모 있으며 또한 체계관리자들에게 쓸모 있는 도구이다.

중복행들을 포함하는 정렬된 파일 dept.lst을 고찰하자.

```
$ cat dept.lst
01|accounts|6213
01|accounts|6213
02|admin|5423
03|marketing|6521
03|marketing|6521
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

uniq는 간단히 매행의 복사판을 하나 꺼내서 표준출력에 그것을 쓴다.

```
$ uniq dept.lst
01|accounts|6213
02|admin|5423
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

uniq는 정렬된 파일을 입력으로 요구하므로 일반적인 수속은 파일을 정렬하고 uniq에 프로쎄스를 관련결하는것이다. 다음의 관흐름도 꼭 같은 출력을 산생시킨다.

```
sort dept.lst | uniq -
```

uniq는 관흐름에서 사용될 때 가장 쓸모 있다. 다른 지령들과 함께 그것을 사용하기전에 먼저 선택항목들을 아는것이 필요하다.

uniq의 선택항목

반복되지 않은 행들을 선택하기(-u)

-u선택항목은 입력에서 유일한 행 즉 반복되지 않은 행들을 선택한다.

```
$ uniq -u dept.lst
02|admin|5423
04|personnel|2365
05|production|9876
06|sales|1006
```

중복행들을 선택하기(-d)

-d(duplicate)선택항목은 반복된 행들에서 한개의 복사행만을 선택한다.

```
$ uniq -d dept.lst
01|accounts|6213
03|marketing|6521
```

발생빈도수를 계수하기(-c)

-c(count)선택항목은 모든 행들의 발생빈도수를 현시한다.

```
$ uniq -c dept.lst
2 01|accounts|6213
1 02|admin|5423
3 03|marketing|6521
1 04|personnel|2365
1 05|production|9876
1 06|sales|1006
```

주의

sort와 마찬가지로 uniq도 출력파일이름을 인수로 받아 들인다. 만일 uniq foo1 foo2지령을 사용하면 uniq는 간단히 foo1을 처리하고 출력과 함께 foo2를 덧쓰기할것이다. 이것은 선택항목을 리용하지 않고 수행되므로(sort에서 -o와 달리) 사용자는 두개의 파일이름과 함께 uniq를 리용할 때 자기가 하고있는것을 알고 있어야 한다.

9.15 행번호붙이기(nl)

nl지령은 행에 번호를 주는 구조를 가지고 있다. pr가 아무런 내용도 없는 행들을 본문에 넣어 주는것과는 달리 nl은 행바꾸기문자외에 어떤 내용을 담고 있는 론리적인 행들만을 넣어 준다(즉 행들에 번호를 준다).

파일 desigx.lst를 고찰하는데 이 파일은 sort의 출력(9.12)에서 생성되였다. nl을 써서 6개 문자만큼 공간을 주고 행에 번호를 붙여 인쇄할수 있다.

```
$ nl desigx.lst
     1 chairman
```

```
2 d.g.m.
3 director
4 g.m.
```

nl은 기정구분문자로서 타브를 사용하지만 -s선택항목을 리용하여 그것을 :으로 변화시킬수 있다. 번호형식의 폭(-w)도 지적할수 있다.

```
$ nl -w1 -s: desigx.lst
1:chairman
2:d.g.m.
3:director
4:g.m.
```

따라서 nl의 번호달기와 형식화능력에 의하여 직위에 대한 표가 얻어 졌다.

주해

nl은 그것이 아무것도 포함하지 않는다면 행에 번호를 달지 않는다. 따라서 이 경우에는 pr가 더 낫다.

9.16 DOS파일과 UNIX파일(dos2unix와 unix2dos)

UNIX와 Windows체계들사이에 파일을 옮겨야 할 경우가 있을수 있다. UNIX파일은 Windows에서 사용되는 DOS형태와 약간 차이나는 구조를 가진다. UNIX에서 모든 행은 행바꾸기문자(8진수 12)로 끝난다. 한편 DOS는 2개의 문자 즉 자리복귀문자(carriage return:8진수 15)와 행바꾸기문자(linefeed)를 사용한다. 아래에 UNIX체계에서 보기한 DOS파일안의 2개 행이 제시되여 있다.

```
Every line contains an extra character^M
DOS files are larger than UNIX files^M
```

UNIX편집기로 이 행들을 포함하고 있는 파일을 편집할 때 매행의 끝에서 ^M([Ctrl-m])을 볼것이다. 이 문자의 8진수값은 무엇인가? 이 두개 행들을 foo파일안에 있다고 가정하고 관흐름을 리용하시오.

```
$ head -1 foo | od -bc
0000000 105 166 145 162 171 040 154 151 156 145 040 143 157 156 164 141
          E   v   e   r   y       l   i   n   e       c   o   n   t   a
0000020 151 156 163 040 141 156 040 145 170 164 162 141 040 143 150 141
          i   n   s       a   n       e   x   t   r   a       c   h   a
0000040 162 141 143 164 145 162 015 012
          r   a   c   t   e   r  \r  \n
```

자리복귀문자는 확장문자렬 \r로 되며 8진수값 015를 가진다. 모든 DOS파일에서 행들은 \r\n로 끝난다. UNIX파일들은 행의 끝에 한개의 \n을 가지고 있다는것을 상기하시오(9.3).

대체로 이 변환은 자동적으로 진행되지만 때때로 자체로 그것을 하여야 한다. 이 목적을 위하여 일부 UNIX체계들은 DOS와 UNIX사이에 파일들을 변환하기 위한 2개의 편의프로그람 즉 dos2unix와 unix2dos를 가진다. 때때로 체계마다 그 수행에서 차이가 있을수 있다.

Solaris체계상에서 UNIX로부터 DOS로 catalog.html파일을 변환하려면 다음의 두 지령들중 어느 하나를 사용할수 있다.

```
unix2dos catalog.html catalogd.html          catalogd.html은 새로운 파일이다
unix2dos catalog.html catalog.html           같은 파일로 변환시킨다
```

일부 체계들은 파일의 내용들이 려과되여 꼭 같은 파일로 씌여 지도록 한개의 파일이름만을 요구한다. 일부는 다시 방향절환을 사용해야 한다.

```
unix2dos catalog.html                같은 파일에 씌여 진다
unix2dos catalog.html > catalogd.html
```

꼭 같은 원리가 dos2unix에도 적용된다. 이제 만일 DOS체계상에서 한개 행으로 보여 지는 파일의 전체 내용들을 발견한다면 unix2dos를 사용하여야 한다. 만일 파일이 UNIX로 옮겨 질 때 매행의 끝에서 ^M을 발견한다면 먼저 dos2unix를 사용하여야 할것이다.

이 두 지령들은 그것들이 자주 완전한 려과기처럼 동작하기때문에 이 장에 포함되였다. 이것은 UNIX파일들이 묶음을 련결할수 있는 방법인데 이때 변환후에 출력을 어떤 한개의 파일로 보관한다.

```
cat *.html | unix2dos > combined.html
```

참고

typescript파일의 모든 행의 끝에 생기는 ^M문자를 지우기 위하여 dos2unix를 사용할수 있다. 이것은 가입대화(3.6)를 기록하는 script가 사용하는 파일이다

9.17 맞춤법검사(spell)

UNIX는 초기에 문서의 준비(preparation)를 위하여 사용되였는데 그것의 모든 판본들은 맞춤법검사(spell-checking)기능을 가지고 있다. spell은 대부분의 UNIX체계들에서 일반적으로 찾아 볼수 있다. 그 지령은 파일을 읽고 틀린것으로 인정되는 모든 단어들의 목록을 만든다.

```
$ spell note.txt
Admintrators
Commuicator
Compter
DAEMON
generalise             미국식맞춤법에서는 "generalize"이다
```

첫 3개는 명백한 오유이고 4번째것은 아니더라도(UNIX에서 현재 리용되는 용어) spell은 그것을 옳은 단어로 인식하지 못한다. 기정적으로 단어들은 미국사전을 참고하여 검사되는데 다른 사전을 리용하려면 -b(British)선택항목을 리용할수 있다.

```
spell -b               영국사전을 사용한다
```

spell은 대화적이지 못하며 Windows사용자가 기대하는것처럼 오유들을 직결식으로 수정할수 없다. 이것은 때때로 관흐름에서 리용될수 있기때문에 유리하다.

```
sort -u foo | spell
```

매행에 한개의 단어를 가진 어떤 파일안에 모든 단어들의 목록을 만들기 위하여 UNIX려과기들을 사용할수 있다. 그러면 단어들의 총수를 보여 주는데 이러한 려과기들을 사용할수 있는가? 실레부분에서 그것을 수행한다.

Linux

대화적인 편집기 ispell을 리용하기

Linux는 더 좋은 맞춤법검사프로그람 ispell을 가지고 있는데 ispell은 대화적인 방식은 물론 려과기로서 조작할수 있다. 대화적으로 리용될 때(파일이름과 함께) 그것은 단어의 근처에 본문을 현시하고 그 행과 단어를 강조한다. 그것은 옳은것을 선택하라는것을 의미하며 다른 선택항목들을 제공한다. 그림 9-2는 ispell에 의해 발생된 화면을 보여 준다.

여기서 단어 browing은 의문을 가지게 한다. 체계는 그 단어를 정확하게 하기 위하여 8개의 가능한 단어들을 선택하도록 제공한다. 만일 목록에서 정확한 단어를 발견하면 번호를 타자하여야 한다. 여기서 사용자가 리용하려고 하는것은 그우에서 보여 주는 WWW에 관한 문맥에 의하여 결정하면 명백히 browsing이였다. 4를 누르면 그 단어를 수정한다.

쓸모 있는 다른 선택항목들이 있다. 그 단어를 받아 들이는데 a를 누를수 있다. 이 경우에 ispell은 후에 같은 파일안에서 그 단어를 만날 때 질문을 하지 않을것이다. 만일 그것이 유효한 기술적인 단어이므로 ispell이 영원히 기억하도록 하고 싶다면 i를 리용하여 자기 소유의 전용사전안에 단어들을 삽입하시오. 만일 리용하여야 할 다른 교체문자렬을 가지고 있다면 r를 리용하시오. q(보관함이 없이)와 x(보관한 다음)로 ispell을 완료한다.

ispell이 -l선택항목과 함께 리용될 때에는 spell과 같이 비대화적으로 동작한다. 그것은 일정한 제한을 준다. 즉 입력은 관흐름으로부터 제공되여야 한다.

```
cat foo | ispell -1                < 에 의한 방향절환이 허락되지 않는다
```

이 지령은 틀린 단어들을 목록으로 만들어 준다. 이 출력에 관하여 sort-u를 실행시키면 spell이 만든것과 꼭 같은 유일목록을 얻을수 있다.

```
browing               File: ux3rd17

*** Messenger, which handles all email and newsgroups
*** Navigator for browing the World Wide Web, and additionally provides the ftp


0: blowing
1: bowing
2: brewing
3: browning
4: browsing
5: crowing
6: growing
7: rowing
. . . .
[SP} <number> R)epl A)ccept I)nsert L)ookup U)ncap Q)uit e(X)it or ? for help
```

그림 9-2. ispell로 편집하기

주해

ispell은 $HOME/.ispell_english(홈등록부안에 있음)파일안에 i가 삽입된 모든 단어들을 보관한다. 오유라고 생각되지 않는 모든 단어들을 추가하기 위하여 편집기로 이 파일을 직접 편집할수 있다.

9.18 려과기의 응용

이제는 기본적인 UNIX려과기들에 대한 지식을 충분히 활용하는 실례부분을 고찰하자. 아직도 취급하여야 할 또 다른 4개의 지령(grep, sed, awk, perl)이 있지만 우리가 이미 알고 있는것들로도 많은 작업을 할수 있다. 이 절에서는 관흐름들에 이 지령들을 리용하여 내용조작과제를 푼다.

9.18.1 발생빈도의 계수

먼저 3번째 마당이 직위를 표현하면서 6개의 마당들을 포함하는 파일 shortlist(9.12)를 고찰하자. 꼭 같은 직위를 가지고 있는 사람의 수를 결정하기 위하여 3단계로 이 작업을 하여야 할것이다.

- 먼저 cut -d"|" -f3 shortlist로 3번째 마당을 자른다.
- 다음에 sort로 정렬시킨다.
- 마지막으로 정렬된 출력상에서 uniq -c를 실행시킨다.

관흐름을 리용하면 이 모든것을 단번에 수행할수 있다.

```
$ cut -d"|" -f3 shortlist | sort | uniq -c
      1 chairman
      1 d.g.m.
      2 director
      1 g.m.
```

이 출력을 만드는것은 수속적인 언어에서 상당한 노력이 요구된다. 후에 per1과 awk가 자기의 자원들을 리용하여 이 상황을 처리하는 방법도 알게 될것이다.

문서제작자들은 사용하는 단어들을 때때로 단어의 발생빈도수와 함께 보고 싶어 한다. 이것을 가능하게 하기 위하여 매 단어는 개별적인 행안에 배치되여야 한다. tr는 모든 공백들과 타브들(8진수 011)을 행바꾸기로 변환함으로써 그것을 할수 있다

```
tr " \011" "\012\012" < foo1                공백은 \040이다
```

\011앞에 공백이 있다. 즉 우리는 그다음에 기호를 현시하지 못할것이다. 만일 단어를 자모문자의 련속적인 묶음으로 정의한다면 첫 tr지령의 출력으로부터 모든 비자모문자들(행바꾸기문자는 제외)을 지우기 위하여 다시 tr를 사용하여야 한다. 이것은 보충(-c)과 지우기(-d)선택항목의 사용을 요구한다.

```
tr " \011" "\012\012" < foo1 | tr -cd "[a-zA-Z\012]"
```

따라서 매 단어가 하나의 개별적인 행우에 놓이는 단어들의 목록이 얻어 졌다. 이제 이 출력을 정렬하고 uniq -c로 그것을 관련결하자.

```
$ tr " \011" "\012\012" < foo1 | tr -cd "[a-zA-Z\012]" | sort | uniq -c
     32 Apache
     18 DNS
```

```
10 Directory
16 FQDN
25 addresses
56 directory
```

단어의 총수를 현시하려면 4개의 지령들을 사용하여야 한다. 반대순서로 목록을 정렬하고 3개 렬로 그것을 인쇄하려면 2개이상의 지령들이 필요할것이다.

```
$ tr " \011" "\012\012" < foo1 | tr -cd "[a-zA-Z\012]" | sort |uniq -c \
> sort -nr | pr -t -3
   56 directory          25 addresses      16 FQDN
   32 Apache             18 DNS            10 Directory
```

이 지령행은 아주 길기때문에 읽기 쉽게 하기 위하여 [Enter]건을 \으로 의미해제하여 그것을 2개 행으로 분할한다.

9.18.2 두 통과암호파일들의 차이점찾아내기

어떤 사용자들의 그룹을 다른 기계로 이동시킬 때 체계관리자에게 필요한 파일은 두 기계의 /etc/passwd이다. 일부 사용자들은 이 기계들에 이미 등록자리들을 가지고 있을수 있지만 일부는 만들어야 한다. 이러한 파일들은 대체로 수백개의 행들을 가지는데 여기서 더 작은 판본들을 가지고 작업한다.

```
$ cat passwd1
joe:!:501:100:joe bloom:/home/henry:/bin/ksh
amadeus:x:506:100::/home/amadeus:/bin/ksh
image:!:502:100:The PPP server account:/home/image:/usr/bin/ksh
bill:!:503:100:Reader's Queries:/home/bill/:/bin/sh
juliet:x:508:100:juliet:/home/julie:/bin/csh
charlie:x:520:100::/home/charlie:/usr/bin/ksh
romeo:x:601:100::/home/romeo:/usr/bin/ksh
ftp:x:602:50:anonymous ftp:/home/ftp:/bin/csh
$ cat passwd2
henry:!:501:100:henry blofeld:/home/henry:/bin/ksh
amadeus:x:506:100::/home/amadeus:/bin/ksh
image:!:502:100:The PPP server account:/home/image:/usr/bin/ksh
bill:!:503:100:Reader's Queries:/home/bill:/bin/sh
juliet:x:508:100:julie andrews:/home/julie:/bin/csh
jenifer:x:510:100:jennifer jones:/home/jennifer:/bin/bash
charlie:x:520:100::/home/charlie:/usr/bin/ksh
romeo:x:601:100::/home/romeo:/usr/bin/ksh
harry:x:602:100:harry's music house:/home/harry:/bin/csh
```

매 파일은 사용자들의 그룹(첫번째 마당)을 제공하는데 우리와 관계되는것은 두번째 파일에는 존재하지 않고 첫번째 파일안에 있는 사용자들의 위치를 정하는것이다. 먼저 passwd1의 첫번째 마당을 잘라

내고 정렬된 출력을 보관한다.

```
cut -f1 -d: passwd1 | sort > temp
```

두번째 파일을 가지고 역시 류사한 련습을 수행할수 있을것이다.

```
cut -d: -f1 passwd2 | sort > temp2
```

이제 comm -23지령을 가지고 2개 파일을 비교하여야 한다. 이 모든 지령들은 려파기이므로 우리는 림시파일 temp2를 만들지 않고 한번의 호출로 일감의 이 부분을 수행하여야 한다.

```
$ cut -d: -f1 passwd2 | sort | comm -23 temp - ; rm temp
ftp
joe
juliet
```

comm -23은 첫번째 파일안에 있는 행들만을 현시하는데 -기호는 sort로부터 출력이 두번째 인수의 위치에서 표준입력으로 제공되였다는것을 담보한다. 목록은 관리자가 useradd지령으로 등록자리들을 만들어야 하는 3명의 사용자들을 보여 준다. 관리자는 능숙한 쉘프로그람작성자이므로 자동적으로 이 일감을 수행하는 script를 사용할것이다.

9.18.3 변경시간과 접근시간을 현시하기

마지막으로 변경시간과 접근시간을 둘 다 보여 주는 파일의 목록을 생성하기 위하여 이 도구들을 사용하자. 이 일감을 위하여 ls -l을 두번 리용하여야 한다. 즉 접근시간을 얻어 내기 위하여 -u선택항목(7.10)과 함께 한번 더 리용하여야 한다. 목록화된 마당들이 여러개의 공백들로 분할되여 있으므로 우리는 cut가 이러한 공백들을 마당으로 인식하도록 하기 위하여 그것들을 압축하여야 할것이다.

우리는 목록의 7개 마당들을 모두 인쇄하지 않고 다만 파일이름, 허가권, 두개의 시간도장들만 인쇄한다. 먼저 개개의 파일안에 있는 변경시간들을 보관하자.

```
ls -l | tr -s ' ' | cut -d" " -f1,6-9 > temp
```

이제 ls -lu지령을 실행한다. 그러나 이때 접근시간만 추출한다(cut -f6-8로). 마지막으로 이 출력을 표준입력이 paste입력의 첫번째 원천지로 되는 paste지령에 관련결한다.

```
$ ls -lu | tr -s ' ' | cut -d" " -f6-8 | paste - temp
     total
Mar 12 08:06    -rwxr-xr-x  Feb 29 19:27 backup.sh
Mar 12 11:38    -rw-r--r--  Mar 12 11:37 foo2
Mar 12 08:06    -rwxr-xr-x  Mar  8 07:51 leapyear.sh
Mar 12 12:17    -rw-r--r--  Mar 12 12:09 passwd1
Mar 12 12:20    -rw-r--r--  Mar 12 12:07 passwd2
Mar 12 12:34    -rw-r--r--  Mar 12 12:34 ux3rd09
```

첫 3개 마당들의 묶음은 접근시간을 보여 주며 파일이름전에 있는 두번째 묶음은 변경시간을 보여준다. ls -l의 첫번째 행은 total행도 보여 준다. 이 행을 제거하려면 이 출력을 또 tail지령에 관련결한다.

```
ls -lu | tr -s ' ' | cut -d" " -f6-8 | paste - temp | tail +2
```

주의

이 실례는 ls -lu가 Linux에서는 다르게 동작하기때문에(표 7-1) 무의미하다. 이 선택항목은 틀림없이 접근시간을 현시하는데 접근시간순서로 목록렬을 현시한다. System V는 -l과 -lu 둘 다 ASCII순서맞추기렬로 목록을 정리한다.

관련결은 UNIX체계의 가장 중요한 측면의 하나를 표현한다. 그것은 복잡한 일감들을 간단한 일감들로 분해하는 려파기들을 결합하는 UNIX원리를 실행한다. UNIX안내서는 매 과제에 대하여 필요한 려파기들의 결합을 가르쳐 주지 못하며 따라서 처음부터 관련결이 복잡하다고 생각하게 만든다. 이 려파기에 대한 지식을 넓히려면 인내성과 상상력이 있어야 한다.

요 약

more는 파일의 출력을 한번에 한 페지씩 보여 주는 페지화프로그람이다. 앞(f)뒤(b)로 이동할수 있으며 패턴(/pattern)에 대한 탐색을 할수 있다. 패턴은 정규식은 물론 간단한 문자렬이 될수 있다. 점으로 마지막지령을 반복할수 있다. 또한 페지화프로그람(v)로부터 vi편집기를 직접 호출할수 있다. wc는 행과 단어들 그리고 문자들을 계수한다. 여러개 파일이름들이 리용될 때 그것은 총 계수값을 인쇄한다.

od는 문자의 8진수값을 현시하며 보이지 않는 문자들을 현시하는데 사용된다. -bc선택항목과 함께 사용될 때 그것은 확장문자렬들인 \f(페지넘기기), \n(행바꾸기), \r(자리복귀)와 \t(타브)를 보여 준다. od는 인쇄할수 없는 문자를 식별하는데 쓰인다.

pr는 머리부와 페지번호들을 인쇄하기 위하여 주로 lp지령과 결합하여 입력을 형식화한다. 출력은 여러개의 렬(-k)로 인쇄될수 있고 두줄공간(-d)으로 구별될수 있으며 어떤 페지번호로부터 시작하도록 설정될수 있다(+k). 모든 머리부와 끄리부를 생략할수도 있다(-t).

쓸모 있는 비교편의프로그람으로서 3개의 파일이 있다. cmp는 처음으로 만나는 차이점위치를 말해준다. 하지만 -l을 써서 문자별로 세세히 볼수 있다.

diff는 사실상 차이나는 행을 보여 주는데 명령묶음을 리용하여 어느 한 파일에 적용하면 그 행은 다른 파일의 대응하는 행으로 변환된다. comm은 공통적인 행과 선택적으로 두 파일중 어느 하나에 고유한 행을 보여 준다.

head는 파일의 시작을 현시하지만 tail은 끝을 현시한다. tail은 어떤 특별한 행으로부터 추출해 내기 위하여 행번호(+k선택항목으로)와 함께 사용될수도 있다. 또한 파일의 증가를 감시하는데 리용될수도 있다(-f). 이 지령의 GNU판본들은 행대신에 문자, 블로크, 다른 단위들을 추출할수 있다(-c).

cut는 자기의 입력으로부터 렬(-c)은 물론 마당(-f)들을 잘라 버린다. 마당번호들은 범위들을 나타내기 위하여 이음표(-)와 함께 증가방향의 번호들과 반점으로 분리된 렬이여야 한다. cut로 잘라 낸것은 paste로 붙일수도 있다. cut와 paste는 타브를 기정구분문자로 리용한다.

sort로 개개의 마당을 정렬시킬수 있으며 또 그 마당안의 렬들도 정렬시킬수 있다. 수자적으로 정렬시킬수 있으며(-n), 정렬순서를 반전할수 있으며(-r), 정렬되였는가를 검사할수 있다(-c). 또한 두개의 정렬된 파일을 통합(-m)하고 중복행을 제거(-u)할수 있다. 정렬순서는 또한 대소문자를 구별하도록 만

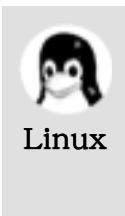
들어 질수 있다(-f).

tr는 문자들을 번역하고 일부 글자들을 변경시키는데 리용될수 있다. 여러번 련속 발생되는것을 압축(-s)할수 있으며 특수한 문자를 지울수 있다(-d). 또한 ASCII 8진수값을 가지고 그것을 사용할수도 있다. 이것은 오직 표준입력을 가지고 동작하는 려파기이다.

uniq는 중복행을 제거하는데 반복되지 않은 행(-u)은 물론 반복되는 행들만(-d)을 목록화하는데 리용될수 있다.

nl은 번호의 폭을 설정할수 있는 론리적인 행들에만 번호를 단다(-w).

UNIX와 DOS파일은 구조에서 차이난다. DOS안에 있는 행들은 자리복귀문자와 행바꾸기문자들로 끝나지만 UNIX행은 오직 행바꾸기문자만을 리용한다. 두개의 편의프로그람 unix2dos와 dos2unix는 변환을 진행한다.

spell은 문서에 대하여 맞춤법을 검사하는데 사용된다. 그것은 틀린 단어들의 목록을 현시한다.

Linux

Linux에 대한 요약

less는 more에 비해 우월한 페지화프로그람이며 vi의 많은 기능을 공유한다. G지령을 사용하여 지적한 행으로 갈수 있다. 뒤로(b) 이동할수 있으며 또한 어떤 패턴에 대하여 방향을 반전하여(?pattern) 탐색할수 있다.

ispell은 spell처럼 단어들을 목록화(-l)할수 있는데 직결방식으로 수정하게 해준다.

시험문제

일부 질문들은 내용이 9.12에서 보여 준 shortlist파일을 리용한다.

1. more를 사용하여 파일을 편집할수 있는가?
2. more로 파일을 보기할 때 문자렬 Internet를 어떻게 탐색하는가?
3. 어떻게 탐색을 반복하는가?
4. 단어란 무엇인가?
5. 파일안에 행번호를 설정하시오.
6. 수자와 자모의 ASCII 8진수값을 어떻게 찾아 낼것인가?
7. 파일에 어떻게 두줄공간을 줄것인가?
8. 만일 cmp foo1 foo2이 그 어떤 출력도 만들지 않는다면 그것은 무엇을 의미하는가?
9. 어떤 사람들에 대하여 정렬된 2개의 목록을 가지고 있다. 이 두 목록에 공통인 이름을 어떻게 찾아낼것인가?
10. echo명령문과 함께 리용된 통보문에서 shortlist안에 있는 어떤 행의 길이를 어떻게 현시하는가?
11. head를 여러개 파일이름과 함께 리용할 때 무슨 현상이 일어 나는가?
12. 반대순서로 행들을 출현시킬수 있는가?
13. 쏘프트웨어설치프로쎄스는 install_log.lst파일에 설치파정을 쓰기한다. 그 파일을 무슨 지령으로 감시할수 있는가?
14. 다음의 지령이 동작하는가?

```
cut -d: -cl -f2 foo
```

15. paste foo[21]로 두 파일을 붙일수 있는가?
16. sort를 리용하여 파일로부터 중복행을 어떻게 제거하는가?
17. shortlist를 출생달에 관하여 정렬하시오.
18. DOS와 UNIX파일사이의 차이점은 무엇인가?
19. 문서의 맞춤법검사를 어떻게 하는가?

련습문제

1. more에서 v를 누르면 vi가 호출된다. 이 지령은 언제 동작하지 않으며 왜 그런가?
2. 홈등록부나무안에 있는 보통파일의 수를 어떻게 계수하는가?
3. 2개의 정렬된 이름목록이 있다. 두 목록에 공통으로 있는 이름들의 수를 어떻게 찾아 내는가?
4. 하나의 공백문자를 포함하는 파일이름을 만드시오. 후에 ls출력으로부터 정말 한개의 공백을 포함하고 있는지 어떻게 확정하는가?
5. 현재등록부안에 있는 모든 파일들을 머리부없이 3개의 렬로 어떻게 현시하겠는가?
6. 두가지 방법으로 파일의 5행부터 10행까지 선택하시오.
7. 첫 10개의 기입항목을 무시한 다음 /etc/passwd로부터 사용자들의 이름을 추출하시오.
8. shortlist로부터 출생년도의 목록을 그 해에 태여난 사람들의 수와 함께 구성하시오.
9. shortlist안에 이름을 거꾸로 정렬시키고 마지막이름다음에 반점을 배치하시오.
10. 만일 체계가 tail의 -r선택항목을 제공하지 않는다면 어떻게 파일을 거꾸로 읽겠는가?
11. shortlist에서 항목들을 선택함으로써 코드목록을 작성하시오. 구분문자로서 :을 사용하시오.
12. 꼭 같은 GUID를 가진 사용자들이 함께 놓이도록 하기 위하여 GUID(기본)와 UID(보조)우에 /etc/passwd파일을 정렬하시오. 더 낮은 UID를 가진 사용자들은 목록에서 더 높은 곳에 배치될것이다.
13. 다음의 두 지령들의 공통점과 차이점은 무엇인가?

```
sort -u foo
uniq foo
```

14. 몇번 나타나는 문자를 어떻게 찾아 낼것인가? 어떤 파일에서 발생하는가?
15. 현재등록부에서 제일 큰 5개 파일들의 목록을 현시하시오.
16. /etc/passwd로부터 UID와 가장 높은 UID를 가진 사용자를 렬거하시오.
17. 한번이상 가입한 사용자들을 렬거하시오.
18. 사용자의 기동파일안에 제공된 기능은 사용자가 가입할 때마다 foo파일에 date지령의 출력을 추가한다. 사용자가 그날에 가입한 회수와 함께 그 날자를 보여 주는 보고서를 어떻게 인쇄할수 있는가?
19. vi를 가지고 매행에서 한개 단어를 가지는 본문을 편집하는 동안 2개 렬로 단어들을 인쇄하기 위하여 화면본문을 어떻게 려과할수 있는가? 단어들이 30부터 130행사이에 위치하고 있다고 가정하시오.
20. UNIX파일을 DOS형식으로 어떻게 변환할수 있는가?

제 10 장. 프로쎄스

앞에서 본바와 같이 UNIX에서는 모든것이 파일이다. 이 장에서는 방향을 바꾸어 모든것을 프로쎄스(process)로서 고찰한다. 독자들은 파일을 가지고 론의를 진행할 때 자기가 공간에 존재하는 느낌을 받았을것이다. 우리는 자기자신과 파일의 위치를 절대위치(뿌리)를 기준으로 하여 알아 낼수 있었다. 이제는 시간차원에서 프로쎄스를 고찰하자. 프로쎄스는 유기체처럼 《생명》을 가지며 다른 프로쎄스들을《낳을》뿐아니라 《죽을》수도 있다. 프로쎄스를 생명체처럼 보고 설명하면 그것을 쉽게 리해시킬수 있다.

UNIX는 다중과제처리체계이므로 한번에 수천개의 프로쎄스를 실행시킬수 있다. 이 장에서는 프로쎄스속성들과 그 속성들을 현시 및 조종하기 위하여 UNIX가 제공하는 도구들을 고찰한다. 대부분의 쉘들은 프로쎄스들을 조종하기 위한 기능을 제공하므로 우리는 이 기능들을 리용하여 필요 없는 프로쎄스들을 제거할뿐아니라 그것들을 정지, 림시정지시킬수 있고 전경과 배경사이에서 이동시킬수 있다. UNIX는 또한 프로쎄스들에 대한 일정작성도구들도 제공한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 프로쎄스의 일반적속성들을 배운다(10.1).
- 프로쎄스생성기구를 리해한다(10.2).
- init가 가입쉘을 어떻게 만드는가를 알아 본다(10.3, 10.4).
- 프로쎄스의 시점으로부터 3가지 형태의 체계지령들을 배운다(10.5).
- ps를 리용하여 프로쎄스속성들을 본다(10.6).
- &와 nohup를 리용하여 배경에서 일감을 실행시킨다(10.7).
- nice를 리용하여 일감우선권을 낮춘다(10.8).
- 신호들이 프로쎄스와 어떻게 통신하는가를 알아 본다(10.9).
- kill을 리용하여 프로쎄스를 제거한다(10.10).
- 일감을 림시정지시켜 전경과 배경사이에서 옮긴다(10.11).
- at와 batch를 리용하여 일감의 실행일정을 작성한다(10.12).
- cron을 리용하여 일감을 주기적으로 실행하기 위한 일정을 작성한다(10.13).
- time을 리용하여 프로그람들의 능력을 비교평가한다(10.14).

10.1 프로쎄스에 대한 리해

프로쎄스란 실행되고 있는 프로그람의 실체(instance)이다. 대부분의 프로그람들은 해당한 프로쎄스를 발생시키며 그 프로쎄스들은 보통 프로그람자체와 같은 이름을 가진다. 실례로 grep지령을 실행하면 grep라는 이름을 가진 프로쎄스가 만들어 진다. 하나의 프로그람이 여러개의 프로쎄스를 발생시키기도 한다. 쉘스크립트를 실행시키거나 관흐름(pipe line)에서 지령묶음들을 실행시킬 때는 한개의 일감이 실행되고 있지만 프로쎄스는 여러개가 실행되고 있다.

UNIX는 다중사용자 및 다중과제처리체계이기때문에 모든 사용자는 한번에 여러개의 프로쎄스를 실

행시킬수 있다. 핵심부(쉘이 아니라)가 종국적으로 이 모든 프로쎄스들의 관리를 담당한다. 그것은 프로쎄스들에 할당될 시간과 우선권을 결정하여 여러개의 프로쎄스들이 CPU자원을 공유할수 있게 한다. 핵심부는 한 프로쎄스가 제한된 시간동안 실행되고 그다음 조종을 다른 프로쎄스에 넘겨 줄수 있도록 하는 기구를 제공한다.

프로쎄스들은 체계의 기억기를 사용한다. 기억기가 가득차면 핵심부는 이 프로쎄스들의 코드와 자료를 **교체구역**(swap area)에로 옮긴다. 교체구역은 보통 그 디스크상의 개별적인 파일체계이다. 프로쎄스가 자기의 시간구간(time slice)을 다시 분배 받으면 그의 영상(image)이 교체구역으로부터 다시 호출되여 또 실행된다. 이것은 1초에 여러번 일어 나며 사용자는 이러한 프로쎄스의 절환을 느끼지 못한다. 체계가 바쁠수록 교체활동은 더 세차진다.

매 프로쎄스는 **프로쎄스식별자**(process identifier:PID)라고 불리우는 수에 의하여 유일적으로 식별된다. PID는 그 프로쎄스가 생성될 때 핵심부에 의하여 할당된다. 프로쎄스는 PID외에 다음의 속성들도 가진다.

- 프로쎄스를 생성한 사용자의 **실제사용자ID**(real user-id). 이때 그 사용자를 그 프로쎄스의 **소유자**(owner)라고 부른다.
- 프로쎄스소유자의 **실제그룹ID**(real group-id). 소유자의 사용자ID와 그룹ID는 둘 다 /etc/passwd안에 저장된다.
- 프로쎄스가 실행되는 우선권. 핵심부의 프로쎄스일정작성프로그람은 이 값을 사용하여 다음번에 실행해야 할 프로쎄스를 결정한다.
- 프로쎄스가 실행된 현재등록부.

실제(real)라는 말을 사용한것은 프로쎄스가 실행되는 동안외에는 마치도 누군가에게 소유되여 있는것처럼 동작하기때문이다. 그것은 또한 모든 프로쎄스가 **유효사용자ID**(effective user-id)도 가지기때문이다. 대부분의 프로쎄스들에서 유효사용자ID는 실제사용자ID와 같은 뜻을 가진다. 우리는 후에(22.4) 그것들이 같지 않게 되는 경우를 고찰할것이다.

주해

프로쎄스는 자기가 실행된 등록부를 기억한다. 이 속성은 프로쎄스가 등록부를 변경할 때 중요하다.

생성과 사멸, 어미와 새끼

MULTLCS로부터의 착상을 빌린다면 프로그람이 실행을 시작할 때 프로쎄스가 **생성된다**(born)고 말하며 프로그람이 동작하고 있는 동안 그것은 계속 《살아 있는다》. 실행이 완성된후에 프로쎄스는 **사멸된다**(die)고 말한다. 파일이 어미를 가지는것과 같이 모든 프로쎄스도 **어미**(parent)를 가진다. 어미그자체는 또 하나의 프로쎄스이며 그로부터 생성된 프로쎄스는 그의 **새끼**(child)라고 한다. 건반으로부터 지령 즉 cat foo을 실행시킬 때 cat지령을 표현하는 프로쎄스가 sh프로쎄스에 의하여 시작된다(Bourne쉘의 경우). sh를 cat의 어미, cat를 sh의새끼라고 한다.

```
at foo
```

모든 프로쎄스들이 어미를 가지므로 UNIX체계에서는 《고아》프로쎄스(어미가 없는 프로쎄스)가 있을수 없다. 모든 프로쎄스의 《조상》을 추적하면 하나의 종국적인 프로쎄스 즉 체계가 기동할 때 설치된 첫 프로쎄스(PID 0)에 도착하게 된다. 이것은 파일체계의 뿌리등록부와 류사하다.

파일 및 등록부와의 류사성은 여기에 국한되지 않는다. 파일과 같이 프로쎄스는 오직 하나의 어미를

가질수 있다. 더우기 등록부가 하나이상의 파일을 가질수 있는것처럼 UNIX의 다중과제처리속성은 프로쎄스가 여러개의 새끼프로쎄스들을 발생(생성)시킬수 있게 한다. 그것을 위한 가장 쉬운 방법은 관흐름을 설치하는것이다. 아래의 지령은 두 지령을 위한 두개의 프로쎄스를 설정한다.

```
cat emp.lst | grep 'director'
```

이 프로쎄스들은 cat와 grep라는 이름을 가지며 둘 다 쉘에 의하여 생성된다.

프로쎄스들이 생명체와 류사하다고는 하였지만 사실 어미가 새끼들에 대하여 가지게 되는 태도는 그렇지 못하다. 어미는 새끼를 발생시킬뿐아니라 그의 사멸을 기다리기도 한다. 새끼프로쎄스가 완성되면 어미에게 사멸을 통지하는 신호를 보낸다. 따라서 조종은 어미에게 되돌려 지며 그러면 어미는 다른 프로쎄스들을 생성할수 있으며 다시 또 그것들이 사멸되기를 기다린다.

10.2 프로쎄스는 어떻게 만들어 지는가

앞에서 언급한바와 같이 프로쎄스는 실행되고 있는 프로그람이다. 파일과 같이 프로쎄스도 CPU가 실행하기 위한 명령문들로서 해석되는 바이트렬이다. 이것은 흔히 실행가능한 프로그람이며 실행을 위한 2진코드와 함께 프로그람이 실행될 때 요구될 자료들을 포함한다. 이 자료는 프로그람코드에서 볼수 있는 변수들과 배렬들로 구성된다.

핵심부에 의하여 현시되는바와 같이 프로쎄스영상(image)은 다른 사용자들이 침범할수 없는 보호된 공간 즉 **사용자주소공간**(user address space)에서 실행된다. 이 주소공간은 몇개의 토막을 가진다.

- **본문토막**(text segment): 프로그람의 실행코드를 포함한다. 여러명의 사용자들이 같은 프로그람을 사용하고 있을수도 있으므로 이 구역은 보통 고정상태로 유지된다.
- **자료토막**(data segment): 프로그람이 사용하는 모든 변수들과 배렬들이 여기에 격납된다. 그것들은 프로그람의 실행과정에 할당된 값을 가지는 변수들과 배렬들을 포함한다.
- **사용자토막**(user segment): 앞에서 언급하였던 모든 프로쎄스속성 즉 UID들과 GUID들, 현재등록부를 포함한다. 핵심부는 이 토막에 저장된 정보를 리용하여 모든 프로쎄스들을 관리한다.

프로쎄스의 만들기에는 3개의 주요한 체계호출들인 fork(), exec(), wait()를 사용하는 3가지 단계가 있다. 프로쎄스만들기에서 그것들이 노는 역할에 대한 지식이 쉘스크립트나 C프로그람들의 오유수정작업을 도와 줄수 있다. 그 3가지 단계를 아래에 설명한다.

- **생성**: UNIX에서 프로쎄스는 fork체계호출을 리용하여 만들어 진다. 이 체계호출은 그것을 호출한 프로쎄스의 사본을 만든다. 실례로 사용자가 프롬프트에 지령을 입력할 때 쉘은 먼저 그자체의 사본을 만든다. 그 영상은 분기생성된(forked) 프로쎄스가 새로운 PID를 얻는다는것을 제외하고는 사실상 호출프로쎄스와 동일하다. 분기생성(forking)기구는 체계안에서 프로쎄스들의 증식을 담당한다.
- **실행**: 다음으로 어미는 생성된 영상을 실행되여야 할 프로그람의 사본에 덧쓴다. 이것은 exec체계호출에 의하여 수행되며 이때 어미는 이 프로쎄스를 실행시킨다(exec)고 말한다. 여기서는 더이상 추가적인 프로쎄스가 생성되지 않는다. 즉 존재하는 프로그람의 본문과 자료구역이 단순히 새로운 프로그람의것으로 교체(덧배치)된다. 이 프로쎄스는 생성된 새끼와 동일한 PID를 가진다.
- **기다리기**: 그다음 어미는 wait체계호출을 실행하여 새끼프로쎄스가 완성되기를 기다린다(wait). 새끼가 실행을 완성하면 어미에게 완료신호를 보낸다. 그러면 어미는 다른 기능들을 계속한다.

프로쎄스는 생성될 때 어미의 환경을 물려 받는다. 즉 그것은 같은 신호들에 응답하며 같은 그룹ID와 사용자ID들, 같은 현재등록부와 우선권 등을 가진다. 다시 말하여 그것은 그 사용자주소공간의 사용자토막안에 있는 거의 모든것을 물려 받는다. 많은 경우에 프로쎄스는 이 파라메터들을 변경하려고 할것이며 그렇게 하는것이 허용된다. 그러나 그때 아주 중요한 문제가 제기된다.

새끼프로쎄스가 자기가 물려 받은 조작환경을 교체한다면 변경된 환경은 그의 어미프로쎄스에서는 쓰이지 못하며 새끼가 소멸되자마자 곧 사라진다. 이것을 리해하고 기억한다면 왜 일부 변수값들이 어디서나 유용하지 않으며 cd지령이 왜 특정한 방식으로 동작하는가를 리해할수 있을것이다.

주해

프로쎄스의 환경파라메터들은 일반적으로 그의 모든 새끼들에게 유용하지만 새끼가 자기의 환경에 만든 변화는 어미에게 전달되지 않는다. 이것은 새끼에게서 정의되거나 재정의된 변수들의 값이 어미에게서는 보이지 않는다는것을 의미한다.

10.3 첫 사용자프로쎄스로서의 가입쉘

사용자가 UNIX체계에 가입할 때 핵심부는 즉시 쉘프로쎄스를 설치한다. 그 프로그람(쉘)은 sh(Bourne shell)일수도 있고 csh(C shell)나 ksh(Korn shell), bash(Bourne Again shell)일수도 있다. 프로쎄스도 그와 같은 이름을 가진다. 프롬프트에 입력하는 임의의 지령은 사실상 쉘프로그람의 표준입력이다. 이 프로쎄스는 체계에서 탈퇴할 때까지 살아 있게 된다.

쉘은 사용자에게 유용한 변수묶음들을 관리한다. 우리는 이미 PATH나 HOME과 같은 많은 변수들을 취급하였다. 가입쉘의 PID는 특수한 《변수》$$에 저장된다. 현재의 쉘에 대한 값을 알아 내려면 아래의 지령을 입력하면 된다.

```
$ echo $$                현재쉘의 PID
659
```

가입쉘의 PID는 사용자가 가입하여 있는 동안 변경되지 않는다. 사용자가 체계에서 탈퇴하였다가 다시 가입할 때 가입쉘은 다른 PID를 할당 받게 될것이다. PID에 관한 지식은 흔히 말단에서 동작을 조종할 때, 특히 무엇이 잘못되였을 때 필요하다.

사용자는 가입쉘로부터 서로 다른 PID를 가지는 여러개의 프로쎄스들을 시작할수 있다. 그렇지만 그것들은 모두 같은 어미 즉 가입쉘을 가질것이다.

10.4 init프로쎄스

가입쉘의 어미는 어느것인가? 일반적으로 그것은 init라는 이름을 가진 체계프로쎄스이다. 이 프로쎄스는 체계의 2번째 프로쎄스로서 PID 1을 가진다. init는 아주 중요한 프로쎄스이며 많은 새끼들을 생성한다. UNIX체계에서 실행되고 있는 많은 프로쎄스들이 init를 자기의 어미로 가지고 있다. 이제 사용자의 쉘을 만들 때 init가 노는 역할을 고찰해 보자.

고전적인 리론에 의하면 체계가 기동하여 다중사용자방식으로 이동할 때 init는 말단에 접속된 모든 포구에서 getty프로쎄스를 분기생성(fork)하여 실행(exec)시킨다. 이 때 getty들은 해당한 말단에 가입프롬프트를 현시하고 잠자기상태로 되여 버린다.

사용자가 가입을 시도할 때 getty는 깨여 나 login프로그람을 실행시켜 입력된 가입이름과 통과암

호를 확인한다. 일반적으로 가입성공시에 login은 사용자의 가입쉘인 쉘프로쎄스를 생성, 실행시킨다. getty와 login은 겹쳐놓기(overlaying)에 의하여 자체로 소멸된다. 쉘프로쎄스를 적재하는 프로쎄스들의 순서를 그림 10-1에 보여 주었다.

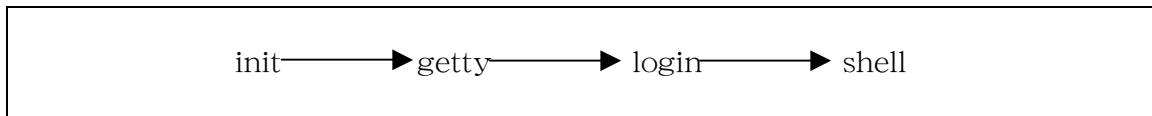



그림 10-1. 쉘을 적재하는 프로쎄스들의 순서

이제 쉘의 유일한 《살아 있는 조상인》 init는 새끼들의 사멸을 기다리면서 잠자기상태로 되여 버린다. 사용자가 체계에서 탈퇴할 때 그의 쉘은 소멸되며 그 종말이 init에 통지된다. init는 그때 깨여 나 다음번 가입을 감시하기 위하여 또하나의 getty를 생성한다. 우리는 체계관리자에 관한 부분에서 이 모든것이 어떻게 init가 사용하는 구성파일인 /etc/inittab에서 실현되는가를 보게 될것이다.

주해

일부 UNIX와 Linux체계들은 다르게 동작한다. 거기서는 login프로쎄스우에 sh프로그람이 덧적재되지 못한다. 이러한 체계들에서는 init가 아니라 오히려 login이 사용자가입쉘의 어미로 된다.

10.5 내부지령과 외부지령

우리는 지령들이 외부지령이거나 내부지령이라는것을 보았다. 쉘은 사실 3가지 형태의 지령들을 인식한다.

- **외부지령**: 가장 일반적으로 사용되는것들은 cat나 ls 등과 같은 UNIX도구들과 프로그람들이다. 쉘은 이러한 매 지령들을 위하여 프로쎄스를 생성하며 그것들을 실행시켜 그의 어미가 된다.
- **쉘스크립트**: 쉘은 또 하나의 쉘(보조쉘)을 생성하는 방법으로 이러한 스크립트들을 실행시키며 그때 그 쉘이 스크립트안에 렬거된 지령들을 실행한다. 보조쉘은 스크립트에 밝혀 진 지령들의 어미로 된다.
- **내부지령**: 그자체가 프로그람작성언어이며 쉘은 여러개의 내장지령들을 가진다. 그들중 cd나 echo와 같은 일부는 프로쎄스를 발생시키지 않고 직접 쉘에 의하여 실행된다. 마찬가지로 명령문 x=5으로 할당된 변수는 프로쎄스를 발생시키지 않는다.

어떤 지령들은 개별적인 외부지령으로 실현하는것이 어렵거나 불가능하기때문에 쉘자체에 내장하는것이 오히려 더 좋다. 우리는 새끼프로쎄스가 현재의 작업등록부를 환경파라메터의 하나로서 자기의 어미로부터 물려 받는다는것은 알고 있다. 이 계승은 cd지령에 중요한 영향을 준다.

cd지령에 관해서는 등록부의 변경을 수행하기 위하여 새끼를 생성하지 말아야 한다. 만일 개별적인 새끼프로쎄스를 통하여 그것을 하였다면 cd가 끝난후에 조종이 어미에게 돌려 지고 원래의 등록부가 회복될것이다. 그러면 등록부를 바꾸는것이 불가능하게 될것이다.

주해

모든 지령들이 프로쎄스를 설정하지는 않는다. pwd나 cd와 같은 쉘의 내장지령들은 프로쎄스를 생성하지 않는다

10.6 프로쎄스의 상태(ps)

파일과 같이 프로쎄스는 많은 속성들을 가지며 그 기능에 대한 옳바른 리해를 위하여 우리는 이 속성들의 일부를 알 필요가 있다. ps(process status)지령이 프로쎄스의 속성을 현시하는데 사용된다. 그것은 파일체계의 ls지령과 짝을 이루는것으로 볼수 있다. 그것은 체계안에 내장된 핵심부에 대한 지식을 가지는 UNIX체계의 몇개의 지령들중의 하나이다. 그것은 핵심부의 자료구조체들이나 프로쎄스표들을 읽어 프로쎄스의 특성을 얻어 낸다.

기정적으로 ps는 그 말단의 사용자에 관계되는 프로쎄스들을 현시한다.

```
$ ps
  PID   TTY        TIME CMD
  659    tty03     00:00:01 sh          ps를 호출한 사용자의 쉘
  684    tty03     00:00:00 ps
```

who -H와 같이 ps도 머리부정보를 발생시킨다. 처음 2개의 렬은 PID와 그 프로쎄스가 관계되는 말단(TTY)을 보여 준다. 일부 프로쎄스들은 말단과 관계되지만 일부는 그렇지 않다. 가입쉘(sh)은 PID 659를 가진다(특정한 변수 $$에 의하여 현시되는것과 같은 번호). TIME은 그 프로쎄스에 의하여 사용된 전체 CPU시간을 보여 준다. 이것은 일반적으로 작은 값이며 대부분의 프로그람들은 보통 자기의 일감을 아주 빨리 완성한다. CMD는 프로쎄스이름을 보여 준다.

이 말단의 사용자는 아무것도 하지 않고 있으며 오직 가입쉘만이 실행되고 있다. 이 출력을 얻기 위하여 ps지령자체가 사용되였기때문에 일부 UNIX체계들은 출력에 ps도 보여 준다(PID 684). 우의것들은 오직 말단 /dev/tty03에 관계되는 지령들뿐이다.

ps의 선택항목

ps는 아주 가변적인 지령이다. 그의 실제적인 출력은 사용되는 하드웨어뿐아니라 UNIX의 판번호에 의존한다. 이 가변성의 특이한 성질은 선택항목들자체가 서로 다른 체계에 대하여 서로 다른 사항을 의미한다는것이다. 선택항목들을 표 10-1에 주었다. 우리는 먼저 System V선택항목들을 고찰할것이다.

표 10-1. ps의 선택항목

선택항목	의 미
-f	매 프로쎄스의 PPID를 렬거
-e	사용자와 체계프로쎄스들을 포함하는 모든 프로쎄스
-u usr	사용자 usr만의 프로쎄스
-a	말단과 련관되지 않은 프로쎄스들을 제외한 모든 사용자들의 프로쎄스
-l	기억기와 관련된 정보의 긴 목록
-t term	말단 term(실례로 tty03)우에서 실행되는 프로쎄스

체계프로쎄스들(-e)

사용자가 생성한 프로쎄스들외에 매 시각 계속 실행되고 있는 체계프로쎄스들이 있다. 대부분은 체계가 기동하는 동안에 발생되는데 일부는 체계가 다중사용자상태로 될 때 기동한다. 그것들을 목록으로

보려면 -e선택항목을 리용하여야 한다.

```
$ ps -e
PID     TTY        TIME  CMD
  0      ?    00:00:00  sched
  1      ?    00:00:01  init                 아주 중요한 프로세스
  2      ?    00:00:00  vhand
  3      ?    00:00:00  bdflush
260      ?    00:00:00  cron                 초시계
282      ?    00:00:00  lpsched              인쇄기데몬
308      ?    00:00:00  rwalld
336      ?    00:00:00  inetd                인터네트데몬
339      ?    00:00:00  routed               경로조종데몬
403      ?    00:00:00  mountd
408      ?    00:00:00  nfsd                 NFS데몬
```

체계프로쎄스들의 특징적인 기능은 그것들이 그 어떤 말단과 전혀 련관되지 않는다는것이다(?로 보여 준것). 이 프로쎄스들의 일부를 **데몬**(daemon)이라고 부르는데 체계에서 중요한 일을 수행한다. 실례로 lpsched데몬은 모든 인쇄동작을 조종한다. TCP/IP망은 inetd데몬이 없으면 실행되지 않는다.

이러한 프로쎄스들은 사용자의 요청이 없이 호출되기때문에 데몬이라고 부르는데 어떤 말단과도 련관되여 있지 않다. 이러한 데몬들의 대부분은 사실 잠 자고 있다가 오직 입력을 받았을 때에만 깨여 난다.그것들중 하나는 이제 할것을 결정 짓는 순간에 자기의 조종파일을 한번 쭉 본다. 다음장들에서 이러한 일부 데몬들과 다른 체계프로쎄스들에 대하여 배운다.이 장에서는 cron데몬을 본다.

완전렬거 (-f)

어떤 프로쎄스의 어미프로쎄스도 보여 주는 세부적인 목록을 얻으려면 -f(full)선택항목을 사용한다.

```
$ ps -f
   UID  PID  PPID  C    STIME     TTY      TIME   CMD
 romeo  659     1  4  18:10:29  tty03  00:00:00   -sh
 romeo  685   659 15  18:26:44  tty03  00:00:00   ps -f
```

여기서 매 프로쎄스의 어미(PPID)와 소유자(UID)를 볼수 있다. 가입쉘(PID659)은 PPID 1을 가지고 있는데 그것은 이 쉘이 이 값을 PID로서 가지고 있는 체계프로쎄스에 의하여 설정되였다는것을 의미한다. 그 체계프로쎄스는 init인데 모든 가입쉘들의 생성자이다. ps의 PPID는 또한 sh의 PID(659)도 보여 준다. 즉 ps는 sh의 새끼이다. 만일 쉘에서 vi지령이 발생한다면 vi의 PPID도 659이다.

-f선택항목이 사용될 때 가입쉘은 일반적으로 지령이름을 이음표(-)와 함께 뒤에 보여 준다. 이것은 실례로 쉘스크립트를 통해서 실행할수 있는 다른 쉘(보조쉘)들로부터 구별하기 쉽게 만들어 져야 한다.

당분간 C머리부는 무시하자. STIME는 프로쎄스가 시작된 시간을 보여 준다. 이때 CMD는 그의 인수들이 다 들어 있는 지령행을 현시한다. 이것은 때때로 리용한 선택항목이 정확히 생각나지 않을 때 유리하다. 자기가 작업하고 있는 파일의 이름을 다른 사람들이 쉽게 알수 있으므로 손해를 줄수 있다고 생각할수도 있다.

사용자프로쎄스들을 현시하기(-u)

체계관리자는 어떤 사용자의 동작을 알기 위하여 -u(user)선택항목으로 사용할것을 요구한다. 실례로 그 사용자가 melinda라고 하자.

```
$ ps -f -u melinda
     UID   PID  PPID  C    STIME    TTY      TIME  CMD
melinda    740   737  0  18:39:59  tty04   00:00:01  -ksh
melinda    761   837  0  18:44:01  tty04   00:00:00  check_number.pl emp.lst
melinda    840   837  0  18:41:09  tty05   00:00:01  -csh
```

melinda는 두 말단에서 가입되였는데 이 선택항목은 두 말단과 련관된 모든 프로쎄스들을 보여 준다. 또한 프로그람들의 완전한 지령행을 현시하기 위하여 -f선택항목을 리용하였다. 그 사용자는 두 말단에서 두가지 형태의 쉘들을 사용하고 있다.

모든 사용자프로쎄스들을 현시하기(-a)

가입쉘들을 비롯하여 모든 사용자들에 의하여 실행되는 모든 프로쎄스들을 현시하기 위하여 -a(all)선택항목을 사용한다.

```
$ ps -a
  PID     TTY        TIME  CMD
  662    tty02    00:00:00  ksh
  705    tty04    00:00:00  sh
 1005    tty01    00:00:00  csh
 1017    tty03    00:00:00  vi
  680    tty02    00:00:00  ksh
 1056    tty02    00:00:00  sort
 1058    tty05    00:00:00  ksh
 1069    tty02    00:00:00  ps
```

현시된 말단이름들로부터 명백히 알수 있는것처럼 5명의 사용자가 여기서 작업하고 있다. 두 일감은 명백한데(vi와 sort) 그 사용자들은 쉘에 대하여 우선권이 서로 다르게 되여 있다. 대부분은 Korn쉘(오늘날 가장 광범히 쓰이는 쉘이다)의 사용자로 되여 있다.

Linux

Linux는 ps지령의 BSD판본을 리용하는데 System V판본에 비하여 뚜렷한 차이를 가진다. Linux에서 ps는 두가지 형태의 선택항목 즉 풀이표를 리용하지 않는 BSD선택항목들과 --(2개의 이음표)를 리용하는 GNU형식의 선택항목을 리용한다. 여기서는 Red Hat Linux를 념두에 두고 론의한다.

체계프로쎄스들(ps ax)

전형적인 Linux체계는 체계프로쎄스들의 묶음을 보여 주는데 그것들을 현시하는데는 -e선택항목보다 오히려 ax선택항목을 리용한다. 아래에 실례를 보여 주었다.

```
$ ps ax
  PID  TTY   STAT  TIME  COMMAND
    1   ?    S     0:14  init                 가입쉘의 어미
    2   ?    SW    0:00  (kflushd)
```

```
   3  ?   SW   0:00  (kpiod)
   4  ?   SW   0:02  (kswapd)
   5  ?   SW   0:00  (mdrecoveryd)
 115  ?   S    0:00  inetd                  인터네트데몬
 125  ?   S    0:00  named                  이름봉사기
 133  ?   SW   0:00  lpd                  인쇄기데몬
 146  ?   SW   0:00  squid -D                 대리봉사기
 148  ?   S    0:00  sendmail: accepting connections on port 25       우편봉사기
 160  ?   SW   0:00  /sbin/mingetty tty6                말단에서 프로쎄스
 161  ?   S    0:00  crond                  체계초시계
 162  ?   S    0:03  httpd                  Web봉사기
```

Linux는 기정적인 많은 망봉사들로 미리 구성되여 있는데 ax선택항목은 그것들모두를 보여 준다. 체계관리자는 사용자들의 접속과 관련하여 이 지령을 리용한다. 만일 사용자가 ftp나 telnet를 할수 없을 때 관리자는 inetd가 실행되고 있는가를 검사해야 한다. 만일 그들이 자기 파일들을 인쇄할수 없는 경우 lpd상태를 검사해야 한다.

완전렬거(ps u)

ps u지령은 SVR4의 ps -f(혹은 ps -l)지령과 비슷하다. 그러나 출력은 더 세부화된다.

```
$ ps u
USER   PID %CPU %MEM  SIZE  RSS  TTY STAT START   TIME COMMAND
sumit  192  0.0  3.5  1892 1088    1 S    20:55   0:00 -bash
sumit  216  0.0  1.9  1576  600    5 S    20:59   0:00 sh /usr/X11R6/bin/sta
sumit  237  0.0  2.9  1908  904    5 S    20:59   0:01 fvwm95
sumit  321  0.0  4.1  1904 1260    1 S    21:02   0:03 vi +12/home/sumit/pr
sumit 3708  0.1 28.4 20732 8728    4 S    09:17   0:04 /opt/netscape/netscap
```

여기서 새로운 렬들을 볼수 있다. 매 지령의 CPU와 기억기사용비률은 각각 %CPU와 %MEM렬에 보여 주었다. 여기서 Web열람기 netscape는 기억공간의 $\frac{1}{4}$이상을 차지한다. 이것은 체계의 성능이 떨어 지는 원인을 찾으려고 할 때 가능한 미해결점을 찾도록 도와 주는 선택항목이다. 프로그람이 기억기에 상주하는 총 용량(KB로)은 SIZE와 RSS렬에 보여 준다.

프로쎄스의 조상을 현시하기(ps f)

PID들과 PPID들을 비교하면서 프로쎄스의 조상을 찾아 내는것은 어려울수 있다. 시각적으로 표현한 프로쎄스나무는 사용자가 찾고 있는 과정을 보여 준다. 다음의것은 Linux의 ps가 f선택항목과 함께 쓰인것이다.

```
$ ps f
  PID TTY STAT   TIME  COMMAND
 3653   4 SW     0:00  (bash)
 3687   4 SW     0:00  \_ (startx)
 3696   4 SW     0:00   |  \_ (xinit)
 3700   4 SW     0:00   |     \_ (sh)
 3701   4 S      0:02   |        \_ kfm
 3708   4 R     24:25   |         |  \_ /opt/netscape/netscape
 3702   4 S      0:01   |         \_ fvwm95
 3969   4 S      0:00   |            \_ xterm -ls
```

여기로부터 startx프로쎄스(PID 3687)는 xinit(PID 3696)의 어미이며 xinit에 기원을 둔 프로쎄스들의 최종어미로 된다는것을 쉽게 알수 있다. 여기서 PPID를 알 필요는 없다. 즉 모든 새끼프로쎄스들을 완료하려면 startx를 완료하여야 한다. 이 선택항목은 SVR4에서는 쓰이지 않지만 프로쎄스의 어미새끼관계를 현시할수 있다는 점에서는 SVR4의 ps -f와 류사하다.

사용자프로쎄스들을 현시하기(ps x)

사용자들에 의하여 실행되는 프로쎄스들을 현시하려면 x선택항목(SVR4에서는 -a)을 리용한다.

```
$ ps x
  PID  TTY  STAT   TIME  COMMAND
  507  tty1  S     0:00  -bash
  593  tty4  S     0:00  -bash
  607  tty3  S     0:00  -bash
  686  tty1  S     0:00  sh /usr/X11R6/bin/startx
  693  tty1  S     0:00  xinit /etc/X11/xinit/xinitrc -- -auth /home/sumit/.xa
  697  tty1  S     0:00  /usr/bin/gnome-session
  706  tty1  S     0:00  gnome-smprxy –sm-client-id default0
  712  tty1  S     0:02  enlightenment –clientId default2
  725  tty1  S     0:00  xscreensaver –no-splash –timeout 20 –nice 10 –lock-mo
  748  tty3  R     0:00  ps x
```

또한 ps f로 본것처럼 여기서 매 프로쎄스들의 상태를 볼수 있다. 오직 하나의 프로그람만이 실행되고 있으며 나머지는 잠 자고 있다.

주해

ps외에 top지령도 더 읽기 쉬운 형태로 CPU사용률을 보여 준다. 이 지령은 Linux에서 가능하며 일부 UNIX체계들은 ps와 같은 출력을 보여 주는데 처음 5개 행들을 가장 흥미 있게 읽을것이다.

```
11:14am  up  3:31,  6 users,  load average: 0.00, 0.00, 0.00
57 processes: 55 sleeping, l running, l zombie, 0 stopped
CPU states:  0.3% user,  0.9% system, 0.0% nice, 98.8% idle
Mem: 30628K av, 29092k used, 1536k free, 17144k shrd, 1376k buff
Swap: 40088k av,  9868k used,  30220k free         10636k cached
```

이로부터 전체적인 정보를 알수 있다. 프로쎄스의 견지에서 사용된 체계기억기의 사용량과 남은량, 그리고 CPU상태를 보여 준다. 기억기의 대부분이 리용되였으나(30628K용량중에서 1536K) CPU는 98.8%에 해당한 시간은 놀고 있다. 이 지령은 체계관리자에게 있어서 매우 쓸모 있는 지령이다.

10.7 배경에서 일감을 실행시키기

다중과제체계는 사용자가 한번에 여러개의 일감을 할수 있도록 한다. **전경**에서는 오직 하나의 일감만을 할수 있으므로 나머지일감들은 **배경**에서 실행되여야 한다. 이것을 실현하는데 두가지 방법이 있다. 즉 쉘의 &연산자와 nohup지령이다. 뒤의것은 일감이 실행되는 동안에도 체계로부터 탈퇴할수 있으나 앞의것은 그렇게 하지 못한다(C쉘과 bash는 제외하고).

10.7.1 체계로부터 탈퇴하지 못한다(&)

&는 배경에서 프로쎄스를 실행시키기 위하여 리용되는 쉘의 연산자이다(즉 전경프로쎄스가 사멸되기를 기다리지 않는다). 지령행이 &로 완료되는 경우 그 지령은 배경에서 실행된다.

```
$ sort -o emp.lst empl.lst &
550                일감의 PID
```

쉘은 즉시 수값 즉 호출된 지령의 PID(550)를 돌려 준다. 프롬프트가 귀환되는데 쉘은 이전의 지령이 아직 끝나지 않았어도 다른 지령을 접수할수 있는 준비가 되여 있다. ps –f 지령을 지금 실행시키면 차이나는 점을 볼수 있다.

```
$ ps -f
    UID  PID  PPID   C     STIME    TTY      TIME  CMD
```

```
romeo  541     1   4  08:22:05  tty02  00:00:00  -sh
romeo  550   541  80  08:22:40  tty02  00:00:03  sort -0 emp.lst emp.lst
romeo  575   541   0  08:26:23  tty03  00:00:00  ps -f
```

sort와 ps는 같은 PPID 즉 쉘의 PID를 가진다. 그 쉘은 두개의 프로쎄스 즉 전경에서 실행되는 프로쎄스와 배경에서 실행되는 프로쎄스를 생성하였다. &를 리용하여 체계부하가 허용하는만큼 많은 일감들을 배경에서 실행시킬수 있다. 쉘은 항상 그것들의 어미프로쎄스로도 된다.

사용하고 있는 쉘에 따라 배경에서 실행되는 일감의 표준출력과 표준오유는 말단으로 갈수도 있고 가지 못할수도 있다. 만일 간다면 두 흐름들의 방향이 절환되였는가를 확인하시오(필요하다면 /dev/null을 리용하여). 이 대책은 C쉘을 사용할 때 취하여야 한다. bash와 함께 이 쉘은 일감을 중단하지 않고 체계로부터 탈퇴하게 해준다.

&기능은 전경에서 작업하고 있는 동안 배경에서 모든 비대화적인 일감들을 실행시키기 위하여 리용된다.

배경에서는 대화적인 일감(실례로 vi편집기 같은것)을 실행할수 없다는것은 명백하다. 만일 그렇게 하면 일감은 림시정지되게 된다. 즉 그러자면 fg(10.11)로 일감을 전경에로 가져다 놓아야 한다. 이러한 제한은 emacs와 xemacs같은 도형편집기에는 적용되지 않는데 X Window체계에서 이러한 편집기들은 자기의 창문들안에서 실행한다.

주의

일감들이 배경에서 실행되고 있을 때 일감들을 너무 많이 실행시키면 CPU의 동작시간이 떨어 질수 있으므로 주의하시오. 그것은 또한 이러한 프로쎄스의 중복으로 정지된 상태에서 실지로 그 기계를 찾고 있는 다른 사용자들에게는 손해로 될수 있다.

10.7.2 안전하게 탈퇴하기(nohup)

배경일감들은 사용자가 체계로부터 탈퇴할 때(C쉘과 bash쉘은 제외하고) 실행을 중지한다. 그것은 쉘이 사멸되기때문에 그런 현상이 일어 난다. 그리고 어미프로쎄스가 완료될 때 그의 새끼프로쎄스 역시 완료된다. UNIX체계는 이 기정기능을 변화시킬수 있게 해준다. nohup지령(no hangup)이 어떤 지령의 앞에 붙으면 사용자가 체계로부터 탈퇴한후에도 프로쎄스가 실행될수 있다. 물론 &도 함께 사용할수 있다.

```
$ nohup sort emp.lst &
586
Sending output to nohup.out
```

쉘은 이번에도 역시 PID를 돌려 주는데 일부 쉘들도 물론 이와 같은 통보문을 현시한다. nohup지령이 이러한 쉘들안에서 실행될 때 nohup은 그 지령의 표준출력을 nohup.out파일로 보낸다. 만일 이 통보문을 받지 않으려면 방향절환(redirection)을 하였는가 혹은 가능한 곳에 출력파일이름을 제공하였는가 확인하시오. 그 지령을 중단시키지 않고 안전하게 체계로부터 탈퇴할수 있다.

또 다른 말단으로부터 nohup을 리용한후 ps지령을 사용할 때(그리고 만일 nohup지령이 아직 완료되지 않았다면) 아주 의의 있는것을 보게 될것이다.

```
$ ps -f -u romeo
   UID   PID  PPID  C     STIME    TTY   TIME  COMMAND
 romeo   586     1 45  14:52:09    01   0:13  sort emp.lst
```

여기서는 체계로부터 탈퇴할 때 쉘이 사멸된다. 그러나 그의 새끼프로쎄스(sort)는 그렇지 않다. 핵

심부는 sort의 PPID를 그 체계의 init프로세스(PID 1) 즉 모든 쉘들의 어미프로쎄스로 단순히 재할당하였다. 사용자가 체계로부터 탈퇴할 때 init는 nohup과 함께 실행되는 어떤 프로쎄스의 어미관계를 넘겨받는다. 이런 방법으로 새끼프로쎄스(sort)는 놔두고 어미프로쎄스(sh)만 제거할수 있다.

어떤 관흐름에서 여러개의 지령을 실행시킨다면 관흐름의 매 지령의 앞에 nohup지령을 리용한다.

```
nohup grep 'director' emp.lst & | nohup sort &
```

C쉘
BASH쉘

배경에서 실행되는 일감들은 표준출력과 표준오유를 말단에로 계속 전송하여야 하는데 화면의 란잡성을 피하기 위하여 방향절환되여야 한다.

일감들은 사용자가 체계로부터 탈퇴한후에도 중단되지 않는다. 이것은 Bourne과 Korn쉘들의 경우에는 다르다. 또한 C쉘에서 nohup지령은 지령의 표준출력을 nohup.out로 보내지 못한다. 그것은 어떤 파일에로 개별적으로 방향절환되여야 한다.

주해

비록 nohup을 리용한다 해도 &기호를 사용해야 한다. 그렇지 않으면 일감은 배경에서 실행되지 않을것이다. 그러나 C쉘의 일부 판본들에서는 nohup자체가 배경에서 어떤 일감을 실행시킨다.

10.8 낮은 우선권을 가진 일감의 실행(nice)

프로쎄스들은 보통 우선순위에 따라 실행된다. 이것은 높은 우선순위일감이 먼저 완성되여야 하므로 그리 좋은것은 못된다. UNIX는 nice지령을 제공하는데 일감의 우선순위를 낮추려면 &연산자와 함께 리용된다. 보다 중요한 일감일수록 체계자원에 대하여 보다 높은 우선권을 가질수 있다.

낮은 우선순위로 일감을 실행하려면 지령이름앞에 앞붙이 nice를 붙여야 한다.

```
nice wc -l uxmanual
```

혹은

```
nice wc -l uxmanual &
```

nice는 C쉘안에 내장되여 있는데 기정적으로 값 4를 가지고 있다. nice값들은 체계게 의존하며 대체로 1부터 19사이에 있다. 지령들은 일반적으로 그 범위안에 있는 nice값(보통 10)으로 실행된다. 보다 높은 nice값일수록 낮은 우선순위로 실행된다. nice는 어떤 프로쎄스의 우선순위를 낮추며 그로 인하여 그 프로쎄스의 nice값은 올라 간다. 또한 nice값을 사용자가 지적할수 있다.

```
nice -5 wc -l uxmanual &              nice값은 5로 증가한다
```

권한이 없는 사용자는 프로쎄스의 우선권을 증가시킬수 없다. 그 권한은 상급사용자(super user)에게 예약되여 있다. nice와 우선권값들은 ps의 -l에 의하여 현시된다.

Linux

nice값은 -20부터 19사이에 있는데 지령들은 0의 nice값으로 실행한다. nice는 어떤 프로쎄스의 nice값을 10으로 높인다. nice는 또한 nice값을 지적하기 위하여 -n선택항목과 함께 리용된다.

10.9 신호

예상외로 오래동안 실행되는 프로그람이 있는 경우 생각이 달라 져 다른것을 실행시키려 한다면 그 프로세스를 끝내야 한다.이것은 일반적으로 새치기건을 누름으로써 수행된다. 그러면 단순히 그 능동프로세스에 완료요구신호를 보낸다. 그 프로세스는 신호를 조종하도록 설계되지 않았다면 완료된다..

신호(signal)란 쉘에 의해 혹은 일부 오유조건에 응답하는 어떤 프로세스에 의해 발생되는 새치기이다. 이 오유조건은 류동소수점은 제외하고 틀리는 명령, 기억기침범 혹은 새치기건일수 있다. 그 사건이 프로세스에 전달된후 프로세스는 아래의 3가지중에서 한가지를 수행하여야 한다.

- **아무것도 하지 않기**: 프로세스는 자기의 동작은 하지 않으며 신호가 그 동작을 하게 한다. 기정동작은 프로세스를 완료하는것이다.
- **신호무시**: 프로세스코드는 신호를 가로 막을수 있어야 하며 그다음 마치 아무일도 일어 나지 않은것처럼 보통프로세스를 계속할수 있어야 한다.
- **신호잡기**: 프로세스는 그 신호를 "잡고" 그 신호에 대해서 미리 결정된 동작을 수행하도록 한다. 과제는 여전히 완료될수 있으며 그렇게 하기전에 일부 림시 파일들을 제거할수 있다.

신호는 어떤 특별한 사건을 나타내는 옹근수로 표현되는데 어떤 체계상에서 쓰이는 일반적인것들은 표 10-2에서 보여 준다. 일부 신호값들은 체계에 의존되므로(일반적인것들이라고 해도) 신호들은 자기의 이름으로 호출되는것이 더 좋다. 자기 기계에 적용할수 있는 완전한 신호목록은 /usr/include/sys/signal.h파일에서 찾아 볼수 있다.

표 10-2. 일반적으로 리용되는 신호들의 목록

번 호	신호이름	기 능
1	SIGHUP	재개 즉 모뎀접속이 중단되는 경우
2	SIGINT	말단새치기 즉 사용자가 새치기건을 누르는 경우
3	SIGQUIT	말단으로부터 탈퇴 즉 프로세스가 주기억덤프파일을 생성하는 경우
9	SIGKILL	무조건적인 제거 즉 신호를 잡을수 없는 경우
15	SIGTERM	kill지령에 리용되는 기정완료신호
24	SIGTSTP	프로세스를 림시정지 즉 사용자가 [Ctrl-z]를 누른 경우

새치기건을 누를 때 SIGINT신호(수값 2)는 현재의 전경프로세스에로 보내진다. 이것은 프로세스가 그 신호를 잡든가 혹은 무시하도록 설계되지 않았다면 프로세스를 제거한다. SIGQUIT(수값 3)는 주기억덤프(현재등록부안에 core라고 이름 지어 진 파일)를 만들어 내도록 프로세스에 지시한다.

그러나 어떤 프로세스가 이러한 모든 신호들을 무시하겠다고 결심하였다 해도 무시 혹은 잡을수 없는 신호 즉 SIGKILL신호(9번 신호)가 있다. 어떤 프로세스가 신호를 프로세스하는 루틴을 가지고 있다해도 이 신호는 최대의 “손상”을 주면서라도 무조건 프로세스를 완료할것이다. 이때 그 프로세스는 완전무결하지 못하게 탈퇴를 하며 완료하기전에 림시파일들을 제거하지 못할것이다.

vi와 같은 프로그람들은 비록 완료신호가 접수되였다 하더라도 하던 작업을 보호하도록 되여 있다. 이 프로그람은 신호들을 조심스럽게 조종하여야 한다.

10.10 프로쎄스의 비정상완료(kill)

kill지령(대부분의 쉘들에 있는 내부지령)으로 프로쎄스를 완료할수 있다. 외부지령 /bin/kill은 쉘이 제거능력을 가지고 있지 않을 때에만 실행된다. 이 지령은 인수로서 하나이상의 PID를 리용하는데 기정적으로 SIGTERM신호(15)를 리용한다. 따라서

```
kill 105                                                     SIGTERM(15)를 리용한다
```

은 PID 105를 가진 일감을 완료한다. 비정상적인 완료를 수행하기 위하여 &연산자는 배경에서 실행되는 프로쎄스의 PID를 현시한다. 만일 PID가 생각나지 않으면 ps지령을 사용하여 그것을 찾아내고 다음 kill을 리용하시오.

만일 배경에서 하나이상의 일감이 실행되고 있다면 단 한나의 kill명령문으로 그것들을 간단히 제거할수 있다. 그것은 바로 kill과 함께 그것들의 PID를 지적하는 방법으로 수행된다.

```
kill 121 122 125 132 138 144
```

만일 이러한 모든 프로쎄스들이 같은 어미를 가진다면 모든 새끼프로쎄스를 제거하기 위하여 어미프로쎄스를 제거할수 있다. 그러나 어떤 지령묶음과 함께 nohup를 사용하고 체계로부터 탈퇴할 때에는 init가 그것들의 어미관계를 획득하므로 어미프로쎄스를 제거할수 없다. 사용자는 init를 제거할수 없기때문에 프로쎄스를 개별적으로 제거하여야 한다.

10.10.1 신호를 지정하기

기정적으로 kill은 프로쎄스를 완료하기 위하여 SIGTERM신호(수값 15)를 사용한다. 일부 프로그람들은 이 새치기를 무시하고 정상적으로 실행을 계속한다. 그 경우에 프로쎄스는 SIGKILL신호(수값 9)로 제거될수 있다. 이 신호는 건누르기에 의해서는 발생되지 않는다. 따라서 kill은 선택항목으로서 신호번호 혹은 이름을 리용하게 한다.

```
kill -9 121                신호번호 9로 제거한다
kill -KILL 121
kill -s 9 121               Solaris는 -s선택항목을 리용한다
```

단순한 kill지령(15번 신호와 함께 리용된 지령)은 가입쉘을 중지시키지 않는다. 아래의 지령중에서 아무것을 사용하여 가입쉘을 제거할수 있다.

```
kill -9 $$                $$은 현재쉘의 PID를 보관한다
kill -9 0                 가입쉘을 포함한 모든 프로쎄스를 제거한다
kill -KILL 0              같다
```

내장된 kill은 또한 일감번호를 받아 들이는데 그것은 쉘의 일감조종기능들(10.11)을 리용할 때만 쓸모 있다. 프로쎄스가 중지되는 과정은 그림 10-2에서 직관적으로 표현되였다.

주해

kill과 함께 신호이름도 사용할수 있다. 실례로 신호 9는 이름이 SIGKILL이다. 따라서 kill -9 121대신에 kill -KILL 121을 리용할수도 있다. 자기의 기계에 적용할수 있는 신호들의 완전한 목록을 보기 위하여 혹은 /usr/include/sys/signal.h 파일을 보기 위하여 kill -1(list)지령을 실행시킬수 있다. Solaris와 Linux체계들에서는 kill -s 9 121 혹은 kill -s KILL 121을 사용해야 한다.

그림 10-2. PID를 발견한후 프로세스를 중지하기

10.10.2 마지막배경일감의 제거

대부분의 쉘들에서 체계변수 $!는 마지막배경일감의 PID 즉 &가 지령에 붙을 때 나타나는 번호를 보관한다. 따라서 PID를 찾기 위하여 ps지령을 사용하지 않고 마지막배경프로쎄스를 제거할수 있다.

```
$ sort -o emp.lst emp.lst &
345
$ kill $!                    sort지령을 제거한다
```

만일 쉘이 일감조종을 지원한다면(대부분의 쉘들은 그렇게 한다.) 하나이상의 일감식별자와 함께 kill을 리용할수 있다. 따라서 kill -9 %1은 신호번호 9로 첫번째 배경일감을 제거하고 kill %s는 이름이 s로 시작하는 프로그람을 완료한다.

C쉘

$!변수는 C쉘에서는 리용할수 없다. 첫번째(혹은 첫번째만) 배경일감을 제거하려면 kill % 혹은 kill %1을 리용하여야 한다.

주해

파일과 마찬가지로 사용자는 실행된 지령에 의해 생성된 프로쎄스들도 소유할수 있다. 또한 소유한 프로쎄스들만 제거할수 있고 다른 사용자들의 프로쎄스들은 제거할수 없다. 더구나 PID 0, 1, 2, 3, 4를 가진 체계프로쎄스들은 이 방식으로 제거할수 없다.

10.11 일감조종

일감이 10분동안에 완성되리라고 기대하였는데 그것이 30분동안 계속된다. 만일 그때 일감을 제거한다면 많은 작업량을 잃을것이다. C쉘 Korn쉘 혹은 bash쉘을 리용하고 있는 경우에는 일감을 조작하는 일감조종기능들을 리용할수 있다. 일부 체계들도 역시 jsh기능(즉 Bourne쉘의 일감조종판본)을 가지고 있다. 이러한 쉘들에서 일감조종은 다음의 기능을 수행한다.

- 일감을 배경에로 이행하기(bg)
- 그 일감을 전경에로 가져오기(fg)
- 능동일감들을 렬거하기(jobs)
- 전경일감을 림시중지하기([Ctrl-z])
- 일감을 제거하기(kill)

이러한 동작들에 대한 지령은 괄호안에 보여 주었다. 어떤 지령을 호출하고 프롬프트가 아직 귀환되지 않았다면 [Ctrl-z]를 눌러 그 일감을 림시중지할수 있다. 그러면 다음과 같은 통보문이 제시된다.

```
[1] + Stopped      spell uxtip02 > uxtip02.spell
```

일감이 아직 완료되지 않았는가를 살펴 보시오. 림시정지만 되였다. 현재전경일감을 배경에로 밀어넣기 위하여 bg지령을 사용할수 있다.

```
$ bg
[1]      spell uxtip02 > uxtip02.spell &
```

행의 끝에 있는 &는 일감이 배경에서 실행되고 있다는것을 지적한다. 따라서 전경일감이 배경일감으로 전환되는데 먼저 [Ctrl-z]를 누르고 다음 bg지령을 쓴다. 또한 아무때나 배경에서 여러개의 일감을 시작할수 있다.

```
$ sort permuted.index > sorted.index &
[2]    530                [2]는 두번째 일감을 지적한다
$ wc -l uxtip?? > word_count &
[3]    540
```

지금 3개의 일감이 실행되고 있으므로 jobs지령으로 그 일감들의 상태를 렬거할수 있다.

```
$ jobs
[3] + Running       wc -l uxtip?? > word_count &
[1] - Running       spell uxtip02 > uxtip02.spell &
[2]   Running       sort permuted.index > sorted.index &
```

fg지령으로 배경일감들을 전경에로 가져 올수 있다. 현재일감(가장 최근의것)을 전경에로 가져 오려면 fg지령을 리용하시오.

```
fg
```

이것은 전경에서 wc지령을 실행시킨다. fg와 bg지령들에는 또한 앞붙이 %기호가 붙은 일감번호, 일감이름 혹은 문자렬이 인수로서 리용될수 있다.

```
fg %1          첫번째 일감을 전경에 가져 온다
fg %sort       sort일감을 전경에 가져 온다
bg %2          두번째 일감을 배경에 보낸다
bg %?perm      문자렬 perm을 포함한 일감을 배경에 보낸다
```

이러한 식별자를 리용하여 kill지령으로 어떤 배경일감을 제거할수 있다.

참고

[ctrl-z]를 눌러 vi 혹은 emacs편집중에 쉘에로 림시탈퇴를 할수 있다. fg를 입력하면 편집방식으로 다시 돌아 간다. 또한 쉘이 일감조종을 지원하는가 안하는가도 결정할수 있다.

여기에서 일감을 림시정지시키기 위하여 [Ctrl-z]를 리용하였는데 이 문자는 기정적으로 stty지령(3.5)에 의해 설정되여 있다. 일감조종을 지원하는 쉘들에서 이 지령을 리용할 때 아래와 같은 출력행을 볼수 있다.

```
start = ^q; stop = ^z; susp = ^z; dsusp = ^y;
```

3번째 마당의 할당은 림시정지문자로 ^z를 보여 주는데 그것은 stty지령이 [ctrl-z]를 표현한것이다. 반드시 필요한것은 아니지만 원한다면 변경시킬수 있다.

10.12 후에 실행하기(at, batch)

UNIX는 지적된 날의 어떤 시간에 실행시키기 위하여 일감의 일정을 작성하는 고급한 기능들을 제공한다. 체계의 부하처리능력이 낮을 때에는 체계부하가 그날에 크게 변화되는 경우 보다 긴급하지 않는 일감들의 일정을 한번에 작성할수 있어야 한다. at와 batch지령은 그러한 일정작성이 가능하도록 만든다. at는 일감이 실행되여야 하는 시간을 인수로서 가진다. 입력은 표준입력에서 제공되여야 한다.

```
$ at 14:08
empawk2.sh
[Ctrl-d]
warning:commands will be executed using /bin/sh
job 951035880.a-574:0 at Sun Feb 20 14:08:00 2000
```

일감은 at대기렬로 가는데 오늘 오후 2시 08분에 스크립트파일 empawk2.sh가 실행될것이다. at는 지정된 실행의 일감번호, 날자, 시간을 보여 준다. 이 일감번호는 1970년이후로부터 계수되여 온 시계의 초수로부터 얻어 진다. 이것은 여러해동안 중복되지 않고 유일번호를 만드는 아주 쓸모 있는 방법이다.

at는 실행되는 스크립트의 이름을 지적하지 못한다. 즉 그것은 사용자가 기억하여야 할것이다. 이 쉘스크립트의 표준출력과 표준오유는 사용자에게 우편으로 보내지는데 그것을 보기 위해 어떤 우편읽기프로그람을 리용할수 있다. 그러나 사용자는 지령의 출력방향을 자체로 결정 짓는것이 더 좋을수 있다.

```
at 15:08
    empawk2.sh > rep.lst
```

또한 파일로부터 지령을 얻으려면 -f선택항목을 리용할수 있다. 그러나 프로그람이 실행될 때 발생

될수 있는 어떤 오유통보문들은 >기호가 없으면 그 사용자에게 우편으로 보내진다. 사용자에게 완성된 일감을 우편으로 보내려면 -m선택항목을 리용하시오

at는 now, noon, midnight, today, tomorrow와 같은 많은 시간형식을 쓴다.

```
at 15                   24시간형식으로 가정
at 5pm
at 3:08
at noon                  오늘 12시에
at now + 1 year                 1년후 현재시간에
at 3:08pm + 1 day                래일 오후 3시 08분에
at 15:08 December 18,2001
at 9am tomorrow
```

일감들은 at -l지령으로 렬거될수 있고 at -r로 제거될수 있다. 공교롭게도 실행되기로 제정된 프로그람의 이름을 찾을수 있는 방법은 없다.

Solaries는 C쉘과 Korn쉘이 실행하는 지령을 얻기 위하여 각각 -c와 -k를 사용한다. 또한 touch에서 허용되는 형식으로 시간을 쓰기 위하여 -t선택항목을 리용한다.

주해

달이름과 요일을 리용할 때는 전부 쓴다든가 혹은 3개 문자로 요약하여야 한다.

10.12.1 batch지령

batch지령은 후에 실행되도록 일감의 일정표를 작성하는데 at와 달리 일감들은 체계부하가 허용되자마자 실행된다. 이 지령은 인수를 가지지 않지만 실행시간을 결정 짓는 내부적인 알고리듬을 리용한다. 이것은 CPU가 요구하는 일감들이 동시에 실행되는 현상을 미리 막는다. batch의 응답은 이외에는 at와 류사하다.

```
$ batch < empawk2.sh
warning: commands will be executed using /bin/sh
job 951018281.b-581:0 at Sun Feb 20 09:14:41 2000
```

batch로 순서화된 일감도 역시 at대기렬로 가는데 at -r지령으로 제거할수도 있다.

10.12.2 at와 batch사용에서의 제한

모든 사용자들은 at와 batch지령들을 리용하지 못할수도 있다. 이 지령에 대한 접근은 /etc/cron.d에 있는 at.allow와 at.deny파일에 의해 제한되며 조종된다. 일부 체계들은 /usr/lib/cron안에 이러한 파일들을 가지고 있는데 Linux는 /etc를 리용한다. 그러나 모든 체계들이 이러한 파일을 반드시 가져야 할 필요는 없다.

기본보안준위가 at.allow에 의해 조종된다. 만일 파일이 있다면 그 파일에 기입된 사용자들만이 at와 batch를 사용할수 있다. 없다면 체계는 이러한 지령의 리용이 금지된 사용자들의 목록에 대한 파일 at.deny를 검사한다. 만일 두 파일이 다 없다면 체계관리자만이 at와 batch를 호출할수 있다. 두 파일은 보통 임의의 사용자들이 읽을수 있지만 오직 체계관리자만이 쓰기할수 있다.

10.13 일감을 주기적으로 실행시키기(cron)

ps -e지령은 실행되고 있는 cron데몬(이름이 “d”로 끝나지 않는 데몬)을 보여 준다. 이 데몬은 매분마다 똑딱거리는 UNIX체계의 초시계이다. 한번만 실행시키는 at, batch와는 달리 cron은 규칙적인 간격으로 프로그람을 실행시킨다.

cron은 대체로 《잠자기》상태에 있다가 매분마다 깨여 나 그 순간에 수행되는 명령들에 대하여 /var/spool/cron/crontabs에 있는 조종파일(crontab파일)을 들여다 본다. 그것들을 실행한후에 다시 잠자리로 돌아 가며 다음번 분에 다시 깨여 난다. 사용자는 이 등록부에 자기의 가입이름으로 이름 지어진 crontab파일을 배치할수 있다. romeo는 crontab지령을 /var/spool/cron/crontabs/romeo파일에 넣어야 한다. 그러나 이 위치는 체계형에 따른다. 매 crontab파일에는 실행일정과 함께 수정되여야 할 지령목록이 들어 있다. crontab내용의 한가지 실례를 그림 10-3에 보여 주었다.

매행은 공백으로 구분되는 6개의 마당으로 이루어 져 있다. 지령행은 마지막마당에 보여 준다. 이 마당들은 지령이 언제 어떻게 실행되는가를 결정한다.

그림 10-3. crontab항목의 구성요소들

cron은 체계를 정합하는 특별한 수를 사용한다. 한조로 되는 수들은 반점으로 구별된다. 첫 5개 마당의 어떤 장소에 *가 리용되면 그 지령은 별표가 놓인 마당에 관하여 주기적으로 실행된다는것을 암시한다.

첫번째 마당(허용값 00부터 59까지)은 지령이 실행되게 될 시간의 분을 지적한다. 범위 00-10은 그 시간의 첫 10분내의 매분마다 실행시킨다. 두번째 마당(17, 즉 오후 5시)은 24시간형식의 시간을 지적한다(허용값 1~24).

세번째 마당(허용값 1~31)은 그 달의 날자를 조종한다. 이 마당은(그림에서 별표) 앞의 두 마당과 함께 읽어 보면 지령이 매일 오후 5시에 시작하여 10분동안의 매분마다 실행되게 될것이라는것을 암시한다. 네번째 마당(3, 6, 9, 12)은 달(허용값 1~12)을 지적한다. 다섯번째 마당(허용값 0~6)은 요일을 가리키는데 일요일은 값이 0이다.

그러면 이 지령의 실행빈도수는 얼마인가? 비록 세번째 마당에서 매일 실행하기 위하여 *를 사용하였지만 다섯번째 마당에서 그 항목을 재정의하여 매주 금요일로 제한하였다. 그리하여 find지령은 매해 3월, 6월, 9월, 12월의 매주 금요일 5시에 시작하여 첫 10분동안의 매분마다 실행되게 될것이다. crontab의 일부 실례들을 표 10-3에 보여 준다.

표 10-3. crontab항목의 실례

crontab항목	의 미
`0,30 * * * *  int_connect.sh`	30분마다 스크립트 int_connect.sh를 실행시킨다
`0 0 * * *  backup.sh`	매일 0시에 스크립트 backup.sh를 실행시킨다
`55 17 * * 4 find / -name core -print`	매주 목요일 17시 55분에 find지령을 실행시킨다
`30 0 10,20 * * du.sh`	매달 10일과 20일 00시 30분에 스크립트 du.sh를 실행시킨다
`00,30 09-17 * * 1-5 mail.sh`	주일(월요일부터 토요일)에 9시부터 17시사이에 반시간마다 스크립트 mail.sh를 실행시킨다

10.13.1 crontab지령

vi나 emacs를 가지고 앞에서 본 형식대로 crontab파일을 만들수 있다. 그러나 cron은 체계가 기동할 때만 그 파일을 읽기하므로 cron이 다시 그 파일을 읽도록 crontab지령을 리용해야 한다.

```
crontab cron.txt                cron.txt파일은 cron지령을 포함하고 있다
```

만일 romeo가 이 지령을 실행시키면 /var/spool/cron/crontabs안에 cron.txt의 내용을 포함하는 파일 romeo가 만들어 진다. 이런 방법으로 여러 사용자들은 자기의 사용자ID들로 이름 지어 진 crontab파일들을 가질수 있다.

또한 -e선택항목과 함께 crontab를 리용함으로써 cron지령들을 입력할수도 있다. crontab는 EDITOR변수에 정의된 편집기(주로 vi)를 호출한다. 거기서 지령을 편집하고 탈퇴한후부터는 그 지령들이 자동적으로 실행일정이 정해 진다.

체계시계가 재설정될 때 cron은 중지되며 다시 시작된다. cron은 결코 체계관리자에 의해서도 제거되지 않는다. 그것은 오래된 파일의 제거, 체계동작에 필요한 자료수집과 같이 주로 《자질그레한》 일을 하는 사람이 리용한다. 이것은 또한 우편물을 보내거나 받는 인터네트우편봉사기에 정기적으로 접속할 때 쓸모 있다.

crontab -l로 crontab파일의 내용을 볼수 있으며 crontab -r로 그것들을 제거할수 있다.

10.13.2 cron에 대한 접근조종

모든 사용자들이 cron기능을 리용할수 있는것은 아니다. at와 batch와 마찬가지로 리용권한은 /etc/cron.d 혹은 /usr/lib/cron안에 있는 두 파일 cron.allow와 cron.cleny에 의해 조종된다. 만일 cron.allow가 있다면 그 파일에 포함된 사용자만이 이 기능을 리용할수 있다. 이 파일이 없다면 사용금지된 사용자들을 확정하기 위하여 cron.deny파일을 검사한다. 두 파일이 다 없는 경우 오직 체계관리자만이 cron리용권한을 가진다.

주의

만일 표준입력을 통한 입력을 제공한다는것을 의미하는 crontab -지령을 리용하다가 그 지령을 중단하기로 결심한다면 [Ctrl-d]가 아니라 말단에 적용된 새치기건으로 완료해야 한다. 만일 그렇게 하는것을 잊어 먹는다면 현재 존재하는 crontab파일로부터 모든 항목들을 제거할것이다.

Linux

cron은 Red Hat에서 var/spool/cron등록부안에 있는 조종파일을 들여다 본다. 그밖에 다른 형식의 명령들이 있는 /etc/crontab파일도 본다. 구체적인것은 cron 혹은 crond문서에서 보시오.

10.14 프로쎄스의 시간측정(time)

어떤 프로그람의 여러 판본들을 가지고 있을 때 그것들이 체계자원을 어떻게 효과적으로 리용하는가 알고 싶을것이다. time지령은 시간이 측정되여야 할 지령을 인수로 리용하는데 프로그람을 실행시키고 그 말단에서의 리용시간을 현시한다. 이것은 프로그람작성자들이 CPU를 최량적으로 리용할수 있도록 자기의 프로그람을 조절할수 있게 한다.

time으로 sort지령이 정렬연산을 진행하는데 걸린 시간을 알아 낸다.

```
$ time sort -o newlist invoice.lst
real   0ml.18s
user   0m0.73s
sys    0m0.38s
```

여기서 real시간은 그 지령이 호출되여 완료될 때까지 경과된 시간을 보여 준다. 이 시간은 여러 프로그람들이 동시에 실행할수 있는 다중사용자체계에서는 좀 차이날수 있다. user시간은 그 프로그람을 실행시키는데 소비된 시간을 보여 주며 sys는 사용자대신 UNIX체계가 작업하는데 리용된 시간을 지적한다.

사용자시간과 체계시간의 합은 CPU시간을 표현한다. 이 합은 일부 시간이 그 체계에서 다른 작업을 수행하는데 소비되므로 실지시간이나 경과된 시간과 반드시 같아야 된다는것은 아니다. 체계에 더많이 적재되면 될수록 더 크게 차이난다.

요 약

프로쎄스란 간단히 말하여 실행되고 있는 프로그람의 실체이다. 프로쎄스는 프로그람이 실행하는한 존재한다. 핵심부는 프로쎄스들이 자기들의 적당한 시간구간(time slice)을 얻어 가지고 필요한 때에 디스크에 교체되여 들어 가는가를 확인한다.

프로쎄스는 프로쎄스ID(PID)에 의하여 식별되는데 매 프로쎄스는 어미를 가지고 있다. 이 어미의 다른 속성들은 웃준위의 사용자ID, 그룹ID, 그 프로쎄스가 실행되는 현재등록부를 포함한다. 한 프로쎄스는 다중새끼프로쎄스를 생성할수 있다.

매 프로쎄스는 코드와 자료로 포함한다. 프로쎄스는 그의 복사판을 만들어 내는 fork체계호출과 함께 그 어미프로쎄스에 의하여 생성된다. 그다음 어미는 그 새끼를 생성하는 exec체계호출에 의하여 복사판을 덧쓰기 한다(exec). 어미는 또한 새끼가 사멸되기를 기다린다.

pwd, cd와 같은 내장된 쉘지령들은 개별적인 프로쎄스를 생성하지 못한다. 쉘은 또한 쉘스크립트를 실행시키기 위하여 보조쉘을 생성한다. 새끼의 환경은 그 어미로부터 일부 파라메터들을 계승하지만 새끼쪽에서의 변경은 어미쪽에 영향을 주지 못한다.

가입쉘은 사용자가 가입하는 동안 실행을 유지하는 사용자프로쎄스이다. 그의 PID는 파라메터 $$로 보관된다. 그 PID는 모든 지령과 쉘로부터 실행되는 스크립트들의 어미이다.

PID 1을 가진 init프로쎄스는 대체로 체계프로쎄스들과 모든 가입쉘들의 어미이다. init는 getty를 생성하고 실행시키며 getty는 login을 생성하고 실행시킨다. 대부분의 체계들에서 login은 쉘을 생성하고 실행시킨다.

ps지령은 프로쎄스들의 속성들을 렬거한다. 사용자는 완전렬거(-f), 사용자의 프로쎄스들(-u), 모든 사용자들(-a), 모든 체계프로쎄스들(-e)을 볼수 있다.

일감은 지령행의 마지막에 &을 붙여 배경에서 실행될수 있다. nohup은 사용자가 체계로부터 탈퇴한 후에도 배경일감이 실행되도록 한다. init는 사용자가 체계로부터 탈퇴할 때 이러한 모든 일감들의 어미관계를 넘겨 받는다. 또한 nice로 어떤 일감의 우선권을 더 낮출수 있다.

프로쎄스에 신호를 보내여 제거할수 있다. 그 프로쎄스는 그 신호를 완료할수 있고 무시할수 있으며 혹은 그밖에 어떤 동작을 할수 있다. kill지령은 어떤 프로쎄스를 완료하기 위한 신호번호와 함께 리용된다. 만일 기정신호번호 15가 동작하지 않는다면 신호 9가 동작(kill -9)할것이다. 대부분의 쉘들에서 마지막배경일감은 kill $!로서 제거될수 있다.

C쉘, bash쉘, Korn쉘은 일감을 조종할수 있다. 일감들은 fg와 bg로 전경과 배경사이에서 이동될수 있으며 [ctrl-z]로 그것들을 림시정지시킬수 있다. fg %2는 두번째 일감을 전경으로 가져 오며 kill %grep는 배경에서 실행하는 grep지령을 제거한다.

at로 한번의 실행을 위하여 일감의 일정을 작성할수 있으며 batch로 체계부하가 허용될 때 그 일감을 실행시킬수 있다. 이 지령의 호출을 조종하기 위하여 체계는 허락된 사용자들의 목록이 있는 at.allow를 검사하는데 그 파일이 없으면 금지된 사용자들의 목록이 있는 at.deny파일을 검사한다. 만일 그 파일도 없으면 오직 체계관리자만이 이러한 봉사를 리용할수 있게 된다.

cron은 일감들이 반복적으로 실행되도록 일정을 작성할수 있게 한다. 그것은 사용자의 crontab파일안에 입력된다. 그 파일은 지령의 실행시간, 빈도수, 완성된 지령을 지적하는 여러개의 마당들을 포함한다. crontab지령은 crontab파일안에 등록항목을 만드는데 리용된다. cron봉사들에로의 접근은 at.allow와 at.deny와 꼭 같은 방식으로 cron.allow와 cron.deny에 의하여 조종된다.

일감들은 time으로서 그 시간이 측정되는데 이것은 프로그람작성자가 프로그람의 여러 판본을 비교하는데 쓸모 있는 도구이다.

Linux

Linux에 대한 요약

Linux에서는 선택항목앞에 풀이표를 리용하지 않는 BSD의 ps지령과 --를 리용하는 GNU형이 쓰인다. ps는 u로 완전렬거(u), 모든 사용자들의 프로쎄스들(x), 모든 체계프로쎄스(ax)들을 볼수 있다. ps는 또한 ps f지령이 리용되면 프로쎄스의 계층도를 보여 준다.

시험문제

1. 프로쎄스를 만드는데 리용되는 두가지 체계호출은 무엇인가?
2. 자기의 가입쉘의 PID를 찾는 가장 쉬운 방법을 무엇인가?
3. 가입쉘은 언제 또 다른 쉘을 생성하는가?
4. 매 말단으로부터 가입요청을 받아 들이는 프로쎄스는 어느것인가?
5. 말단과 련관되지 않은 일부 프로쎄스들의 이름을 말하시오.
6. ps출력으로부터 머리부행을 어떻게 제거할수 있는가?
7. ps -f는 왜 사용자의 내부적인 동작에 불법침입하는것으로 생각되는가?
8. 누구나 다 프로쎄스의 우선권을 높일수 있는가?
9. 프로쎄스가 제거되였는가를 확인하려면 어느 선택항목을 kill에 리용해야 하는가?
10. 누구나 다 다른 사용자들의 프로쎄스들을 제거할수 있는가?
11. 신호목록을 어떻게 현시하는가?
12. jobs지령이 통보문 jobs:not found를 발생시켰다. 이 통보문은 대체로 어떤 때에 발생하는가?
13. vi로 어떤 파일을 편집하다가 쉘로 림시탈퇴하려고 한다. 그렇게 하려면 어떻게 하여야 하며 vi로 다시 돌아 가려면 어떻게 하여야 하는가(C쉘, Korn쉘 bash중에 어느 하나를 사용하고 있다고 가정하시오)?
14. at와 batch로 실행되여야 할 일감의 이름을 어떻게 찾는가?
15. 래일 오후 8시에 스크립트 dial.sh가 실행되도록 at지령을 작성하시오.
16. crontab 내용을 만들기 위하여 crontab지령을 호출하고 변경시켰다. crontab가 현재 바라는 표준입력을 어떻게 완료하는가?
17. 두 프로그람 foo1과 foo2의 본질을 어떻게 비교해 볼수 있는가?

련습문제

1. 프로쎄스의 조종에서 노는 교체구역의 역할은 무엇인가?
2. 만일 두 사용자가 같은 프로그람을 실행시킨다면 기억기가 2배로 요구되는가?
3. foo등록부를 만들고 아래의 두 지령으로 쉘스크립트를 채워 넣었다. 이 스크립트를 실행시킬 때 무슨 현상이 일어 나며 왜 그런가?

```
cd foo; pwd
```

4. 어느 프로쎄스가 새끼의 최대번호를 가질수 있다고 생각하는가. 그것의 PID는 얼마인가. 그것의 새끼를 두가지 종류로 나눌수 있는가?
5. 사용자가 체계로부터 탈퇴할 때 init는 무엇을 하는가?

6. login프로그람의 역할은 무엇인가?
7. 아래의 관흐름이 어떤 쉘스크립트로부터 실행된다면 어느것이 이 모든 프로쎄스들의 어미와 어미웃준위인가?

```
nl foo | sort -nr cut -f2-
```

8. 사용자 romeo에 의하여 실행되는 어떤 프로쎄스의 완성된 지령행을 어떻게 찾는가?
9. ps -e와 ps -a의 차이점은 무엇인가?
10. 데몬의 기본속성은 무엇인가?
11. 어떤 파일을 편집하고 싶은데 ps지령으로 그 파일이름을 보여 주고 싶지 않는 경우 vi와 emacs로 그것을 어떻게 실현하는가?
12. 인쇄할수 없는 경우 ps출력에서 어느 프로쎄스를 찾을것인가?
13. kill -9 0은 어느 프로쎄스를 완료하는가?
14. 마지막배경일감의 PID를 모르는 상태에서 그것을 어떻게 제거해야 하는가. 그것은 모든 쉘들에서 실행하는가?
15. &로 실행된 프로쎄스와 nohup로 실행된 프로쎄스와의 차이점은 무엇인가?
16. 다음과 같은 지령이 실행되는가?

```
nohup compute.sh
```

17. 만일 nohup로 프로그람을 실행시키고 체계로부터 탈퇴한다면 프로쎄스의 어미관계에서 어떤 현상이 일어 나는가?
18. 개별적인 프로쎄스를 요구하지 않는 일부 지령들의 이름을 말하시오.
19. exit지령은 무엇을 하는가?
20. 신호번호 2와 9는 어떻게 발생되는가? 그것들의 이름은 무엇이며 기능적인 측면에서 어떻게 차이나는가?
21. ps지령은 같은 말단상에서 실행되고 있는 emacs지령을 보여 준다. 사용자는 emacs화면은 보지 못하고 다만 쉘프롬프트만 보는 상태이다. 언제 그 화면이 나타나며 거기서 무엇을 할수 있는가?
22. 다음의 crontab항목들을 해석하시오.

```
(1) * * * * * dial.sh
(2) 30 21 * * * find /tmp /usr/tmp -atime +30 -exec rm -f {} \;
```

23. 다음의 crotab항목은 일부 오유를 가지고 있다. 그것은 무엇인가?

```
00-60 22-24 30 2 * find.sh
```

24. 매주 월요일, 수요일, 금요일 오전 9시부터 오후 9시사이 15분마다 int_connect.sh 스크립트를 실행시키는 crontab항목을 작성하시오.
25. 관리자는 대부분의 사용자들이 at와 cron을 리용할수 있게 하려고 한다. 쉽게 하려면 무엇을 변경시켜야 하는가?
26. 체계관리자가 at와 cron의 어느 하나만을 사용하도록 하자면 어떻게 해야 하는가?

제 11 장. TCP/IP망작업도구

초기에는 콤퓨터들이 독립적으로 떨어 져 있었다. 후에 사람들은 독립적으로 떨어 져 있는 콤퓨터들은 아무런 의의도 가지지 않는다는것을 깨달았다. 즉 콤퓨터들은 다른 콤퓨터와 대화를 진행해야 하였다. 초기의 TCP/IP는 독립적으로 떨어 져 있는 망들이 약간이나마 의의를 가진다는 생각에 뿌리를 두고 있다. 망들도 역시 리용하는 하드웨어나 쏘프트웨어 그리고 조작체계에 관계없이 다른 망들과 통신하는것이 필요하였다. 그런데 문제로 되는것은 서로 다른 이 모든 종류를 다 포함한 통신규약의 묶음이다. 이 통신규약의 묶음을 TCP/IP라고 한다.

TCP/IP의 특징은 기술의 발전이 그 기술을 발전시키는 도구들의 발명과 함께 이루어 진다는것이다. 독립적으로 떨어 져 있는 대다수의 UNIX응용프로그람들도 망에서 작업할수 있도록 달라 졌다. 오늘 TCP/IP망작업도구들은 모든 UNIX체계의 핵심부에 내장되여 있다. TCP/IP는 또한 인터네트의 통신규약이다. 우리가 이 장에서 론의하는 모든 TCP/IP도구들은 인터네트상에서도 동작한다.

이 장에서는 다음과 같은 내용들을 학습ㄱ하게 된다.

- 파케트절환망이 동작하는 방식을 리해한다(11.1).
- 주콤퓨터이름을 /etc/hosts 파일을 리용하여 IP주소들로 변환시키는 방법을 배운다(11.1.2).
- 인터네트상에서 주콤퓨터이름들을 완전지정령역이름(FQDN)들로 교체하는 방법을 배운다(11.1.3).
- talk를 리용하여 다른 사용자와 실시간적이며 본문방식에 기초한 대화를 진행한다(11.2).
- finger로 원격체계우의 사용자들에 대한 세부들을 현시한다(11.4).
- telnet와 rlogin을 리용하여 원격기계에 가입한다(11.5, 11.6).
- ftp와 rcp를 리용하여 두 기계사이에 파일을 전송한다(11.7, 11.8).
- rsh로 지령을 원격실행시킨다(11.9).
- rlogin, rcp, rsh를 리용할수 있게 하는 방법들을 배운다(11.10).

11.1 TCP/IP의 기초개념

TCP/IP는 망통신규약묶음이다. 이 통신규약들은 망에서 다른 기계와의 통신에 응답해야 하는 규칙들을 정의하고 있다. TCP/IP는 Transmission Control Protocol/ Internet Protocol 의 략자인데 여러개의 통신규약(그중에서 TCP와 IP는 가장 중요한 통신규약들이다.)들을 묶은것이므로 이름의 의미가 어느 정도 맞지 않는다.

TCP/IP의 특징:

- 제공자와 기계, 조작체계의 종류에 무관계한것
- 다중파케트들로 자료를 전송하는것
- 자료가 하나 혹은 그이상의 망들을 통과했을 경우 다른 경로(route)들에로 즉시 자료방향을 되돌리는 능력

- 완전한 오유조종기능으로 하여 전송믿음도가 100%인것

전화체계와는 달리 TCP/IP는 **파케트절환**체계이다. 파케트절환망에서 전송자와 수신자는 항상 련결되여 있지 않다. 오히려 자료는 파케트들로 끊어 지고 매 파케트는 머리부(**봉투**)와 함께 제공된다. 이 머리부는 렬번호와 검사합(checksum)을 가지고 있는데 파케트에서 정보가 정확한가를 결정한다. 파케트들은 봉투(envelope)안에 넣어 지고 송신자와 수신자의 주소는 그 봉투에 씌여 진다. 그리고 그 파케트들은 다음과 같은 방법으로 보내진다.

파케트들은 인터네트와 같이 거대한 망을 따라 전송되므로 그 파케트들은 모든 곳에서 경로기들을 만난다. **경로기**(router)란 봉투주소를 보고 매 파케트들이 목적지까지 짧은 시간동안에 전송되게 하는 가장 효률적인 경로를 결정하는 특수한 콤퓨터나 지능적인 장치를 말한다. 파케트들은 망에 끊임없이 적재되기때문에 여러개의 경로를 따라 움직이며 복잡하게 뒤얽혀 져 목적지에 도착한다. 그 파케트들은 그 안에 제공된 정보로부터 정확한 순서로 다시 재조립된다.

파케트들을 조립하기전에 매 파케트의 검사합은 그 파케트에 보내진 수에 의하여 계산되고 검사된다. 만일 검사합이 맞지 않으면 그 파케트는 정확하지 못하며 재전송되여야 한다. 모든 파케트들이 말끔히 접수되면 파케트들은 조립되며 그것들의 머리부는 없어 진다. 그리고 자료는 초기의 형식대로 응용프로그람에 보내진다.

이 기능들은 망에서 실행되는 응용프로그람(ftp나 telnet와 같은)안에 내장되여 있지 않다. 그것들은 기계들에 리용되는 조작체계나 하드웨어세부들에 완전히 독립이다.

11.1.1 주콤퓨터이름과 IP주소

망에서 어떤 콤퓨터를 **주콤퓨터**(host)라고 하며 주콤퓨터는 **주콤퓨터이름**(hostname)을 가지고 있다. 이 이름은 망에서 유일하다. 주콤퓨터이름을 알기 위한 가장 쉬운 방법은 hostname지령을 리용하는 것이다.

```
$ hostname
saturn.planets.com
```

이 이름은 인터네트상에서 주콤퓨터들이 리용하는 이름인데 사실 첫번째 단어가 주콤퓨터이름을 의미한다. planets.com은 주콤퓨터가 속하는 령역의 이름이다. 어떤 체계들은 hostname 지령의 출력에 령역이름을 쓰지 않는데 만일 체계가 령역이름을 쓰는 경우에는 hostname지령과 함께 -s선택항목을 리용하여 빼버릴수 있다.

```
$ hostname -s
saturn
```

주콤퓨터이름외에 망에서 매 주콤퓨터는 다른 기계들이 그 주콤퓨터와 통신하는데 리용되는 **IP주소**를 가지고 있다. 이 주소는 형태적으로 볼수 있는바와 같이 점으로 구별되는 련속적인 수들로 이루어 져 있다.

```
192.168.0.1
```

IP주소도 망관련하드웨어와 독립이며 체계의 기동스크립트들에 의해서 기동시에 설정된다. TCP/IP 응용프로그람들은 주콤퓨터이름은 물론 IP주소로 주콤퓨터를 주소화할수 있다.

```
ftp 192.168.0.1
```

```
telnet saturn
```

번호는 기억하기가 어려우므로 사용자들은 IP주소보다 주콤퓨터이름들로 기계들을 호출하는것을 더 좋아 한다. 그렇지만 최종적으로 기계들은 오직 IP주소들만 리해하며 주콤퓨터이름으로부터 IP주소로 변환하는것은 두 기계가 서로 련결되기전에 되여야 한다.

주해

date지령과 마찬가지로 hostname지령들은 체계관리자에 의하여 주콤퓨터이름을 설정하기 위한 인수와 함께 리용할수 있다.

11.1.2 주콤퓨터이름을 IP주소에 대응시키기(/etc/hosts)

소규모적인 망에서 이름-주소넘기기는 그 망의 주콤퓨터에 있는 /etc/hosts파일안에 보관된다. 이 파일을 흔히 **주콤퓨터파일**(hosts file)이라고 한다. 이 자료기지들은 응용프로그람이 이름을 보고 그에 대응하는 IP주소를 찾도록 해준다. 그것들을 보여 주는 실례파일을 보자.

```
$ cat /etc/hosts
127.0.0.1        local host
192.168.0.1      saturn
192.168.0.2      uranus
192.168.0.3      jupiter j2             허락된 별명들
```

망의 매 주콤퓨터에 대해서 이 표는 IP주소를 련관된 주콤퓨터이름으로 넘기는 행을 담고 있다. 마지막행은 TCP/IP가 별명을 리용할수 있게 해준다는것을 보여 준다. 이것은 telnet 192.168.0.3대신에 간단히 telnet jupiter를 리용할수 있게 해준다. 그리고 이것도 지내 복잡한것 같으면 telnet j2로 리용할수도 있다.

참고

TCP/IP봉사를 리용하겠다고 망에 접속할 필요는 없다. **국부주콤퓨터**(localhost)이름이나 IP주소 127.0.0.1을 리용해서 자기 기계에 접근할수 있다. 사용자는 언제나 모든 /etc/hosts파일안에서 이 이름을 IP주소로 넘기는 행을 찾아 볼수 있다. 이것은 사용자의 기계가 망에 존재하지 않아도 지원된다.

주해

telnet와 ftp기능을 제공하는 Windows상에서도 주콤퓨터파일이 존재한다. 이 파일의 위치는 대체로 \WINDOWS(WIN95/98) 혹은 \WINDOWS\SYSTEM32\DRIVERS\ETC(Win 2000/NT)이다. 만일 Windows주기계로부터 어떤 주콤퓨터이름을 가진 UNIX기계에 접근한다면 접근자는 이 파일을 편집해야 한다.

11.1.3 령역이름체계

대규모의 망에서 매 기계안에 있는 주콤퓨터파일을 관리하는것은 망관리자에게 있어서 처리하기 어려운 과제이다. 그러나 다른 체계 즉 **령역이름체계**(Domain Name System:DNS)를 리용하는 인터네트상에서는 가능한 과제이다. DNS리론은 주콤퓨터를 간단한 이름으로서가 아니라 하나 혹은 그이상의 **령역**들의 구성체로서 취급한다. 령역들은 계층적인 형태를 가지고 국부적으로 관리된다. 매 령역의 자료는 이미 다른 모든 령역들에 의하여 리용될수 있게 만들어 진다.

령역이름체계에서는 인터네트**이름공간**(령역나무)이 UNIX파일체계와 함께 취급된다. 마찬가지로 이것도 점(.)이라고 하는(이름이 없는) 뿌리령역(root domain)을 가지고 있다. 이 뿌리령역은 그밑에 몇개

의 뿌리준위령역(root-level domain) 혹은 웃준위령역(top-level domain)을 가지고 있다. edu와 com은 웃준위령역의 실례이다. 이 령역들은 등록부가 보조등록부를 가지고 있는것처럼 보조준위령역(second-level domain) 혹은 부분령역(sub-domain)들을 가지고 있다. 보조준위령역들은 또 부분령역들을 가지며 이것들도 여러개의 준위들로 내려 간다.

실례로 binhamton은 edu의 보조준위령역이며 자기밑에 cs부분령역을 가진다. 그러므로 host ralph는 다음과 같이 유일적으로 서술될수 있다.

```
ralph.cs.binghamton.edu.                (뒤에 점이 있다)
```

이것은 주콤퓨터 ralph의 **완전지정령역이름**(fully qualified domain name:**FQDN**)을 표현한다. 이것은 파일의 절대경로와 비슷하다. 또 다른 싸이트(어떤 기관안에 있는 주콤퓨터들의 모임)도 주콤퓨터이름 ralph나 부분령역이름 cs를 가질수 있지만 그의 FQDN은 언제나 다르다. 이 이름은 인터네트상에서 유일하다.

령역이름들은 대소문자를 구별하지 않는다. Cs.BinghAmtOn.eDU를 리용한것과 같이 CS.BINGHAMTON.EDU를 리용할수 있다. 기관은 짧은 FQDN을 가져야 한다. 즉 기억하기 쉽고 기관을 쉽게 식별할수 있어야 한다.

최종적으로 FQDN은 BIND(인터네트상에서 DNS를 실현하기 위하여 일반적으로 리용되는 쏘프트웨어)를 리용하여 대응하는 IP주소로 넘겨 져야 한다.이 이름봉사는 분석(resolution)을 위하여 각이한 기계들의 령역이름을 참조하기 위한 규칙에 따른다.

그 분석과정과 DNS실현을 관리하는 리론은 제24장에서 론의된다.

주해

전자우편주소, 령역이름, IRC(Internet Relay Chat:인터네트중계담화)는 대소문자를 구별하지 않는 UNIX의 유일한 실체들이다.

11.1.4 데몬과 포구, 소케트

TCP/IP와 인터네트는 망으로 련결된 두 콤퓨터사이의 작업을 분할하는 **의뢰기-봉사기**원리에 기초하고 있다. 어떤 ftp응용프로그람은 봉사기와 의뢰기요소로 분리된다. Web봉사기의 일감은 HTML문서를 보내는것인데 의뢰기쏘프트웨어(열람기)는 그 문서를 사용자화면에 보여 준다.

Linux에서 봉사프로그람들은 **데몬**(daemoen)이라고 부르는데 그에 대해서는 ps지령(10.6)의 출력에서 대체로 보았다. 그것들은 어떠한 말단과도 련관이 없으며 언제나 의뢰기들로부터 입력되는 배경에서만 동작한다. httpd데몬들은 필요한 Web페지를 받아 들인다. sendmail은 우편물을 처리하는 데몬이다.

ftp파케트는 자기가 Web열람기에로가 아니라 다른 말단의 또 다른 ftp 응용프로그람에 련결되여야 한다는것을 어떻게 아는가? ftp파케트에 IP주소를 지적하는것으로써는 불충분하다. 개별적인 **포구번호**도 파케트에 포함되여야 한다. 이 번호는 언제나 ftp봉사와 관련된다. 이로부터 그 파케트는 ftp봉사기에로 도달한다. 포구번호들은 사무실의 확장번호와 같다. 사무실전화번호를 지적하는것으로써는 불충분하다. 즉 한 사람이 다른 사람에게 파케트를 보내려고 해도 확장번호를 알아야 한다.

데몬들은 그들에게 할당된 포구에서 요청들을 접수한다. sendmail은 포구 25에서, ftp는 포구 21에서 그리고 telnet는 포구 23에서 요청을 접수한다. 그러나 상대켠의 의뢰기들은 임의의 포구번호를 사용한다. 실례로 ftp 는 의뢰기쪽에서 어떤 임의의 수를 리용하여 련결하지만 봉사기쪽에서는 오직 포구번호 21을 지적할것이다.

봉사기프로그람들에 리용되는 이 포구번호들은 /etc/services 안에 기입된다. 우에서 본 대표적인 봉사기들은 이 파일에서 다음과 같은 내용을 가진다.

```
ftp        21/tcp
telnet     23/tcp
smtp       25/tcp        mail
www        80/tcp        http     #WorldWideWeb HTTP
pop3       110/tcp       # POP version 3
```

모든 파케트들은 4개의 번호들로 이루어 진 모임을 가지고 있다. 즉 량쪽의 IP주소들과 TCP포구번호들이다. 이 모임을 **소케트**(socket) 때로는 **접속**(connection)이라고 부른다. 임의의 두개의 소케트들은 같은 번호모임을 가질수 없다. 만일 두명의 사용자가 같은 봉사기에 같은 IP주소로 Web페지를 요구한다고 해도 그 소케트들은 의뢰기쪽에서 우연발생적인 포구번호를 사용하기때문에 서로 다르다.

TCP의 이 기능적인 측면들에 대한 지식은 아래에서 보게 되는 망도구들에 대한 설명에서도 계속 리용된다. ftp와 telnet는 인터네트상에서도 광범히 리용된다. 때때로 실례파일 /etc/hosts에 기입된 IP주소들과 주콤퓨터이름들도 리용된다.

11.2 실시간대화(talk)

talk는 UNIX의 모든 판본에서 가능한 대중적인 망통신프로그람이다. talk는 사용자가 현재 등록가입된 어떤 사람과 두가지 방식으로 통신하게 한다. 즉 talk는 한 사용자가 입력한 본문이 다른 사용자의 말단에 현시되는 본문방식의 대화이다. talk는 write지령을 필요 없게 만들며 IRC(인터네트중계담화)의 《조상》이다.

talk는 같은 주콤퓨터나 원격주콤퓨터의 다른 사용자와 통보문을 교환하는데 리용될수 있다. 아래에서는 henry가 사용자 charlie와 통신하기 위하여 talk를 호출한다.

```
talk charlie              같은 주콤퓨터상의 사용자
talk charlie@saturn       원격주콤퓨터상의 사용자
```

주소화의 두번째 형식을 주목해서 보시오. 이것은 어떤 사람에 대하여 전자우편통보문을 주소화하는데도 리용되는 형식이다. 여기서 주소는 두개의 요소로 이루어 진다. 즉 @기호에 의해 분리된 사용자이름과 주콤퓨터이름이다.

charlie가 등록가입되여 있다면 henry는 그와 대화하고 싶다는것을 통지하게 될것이다. charlie는 자기의 화면상에 현시되는것을 볼수 있다.

```
Message from Talk_Daemon@uranus at 10:20 ...
talk: connection requested by henry@uranus
talk: respond with: talk henry@uranus
```

지금 charlie는 talk를 호출하여 자기의 통보문을 입력하는 방법으로 응답해야 한다. charlie는 henry가 같은 주콤퓨터에 있는가 아니면 다른 주콤퓨터에 있는가에 따라 다음의 지령중의 하나를 리용할것이다.

```
talk henry               자기 주콤퓨터상의 henry
```

```
talk henry@uranus            다른 주콤퓨터상의 henry
```

이 지령이 일단 charlie에 의해 입력되고 henry에 의해 접수되면 두 사용자사이에는 접속(connection)이 설정된다. talk는 그때 두 기계에 수평으로 분할된 창문을 보여 준다(그림 11-1).

```
[Connect established]
What is that you wanted to say?

--------------------------------------------------------------------------------

Was just wondering whether you received the message from jim.
```

그림 11-1. talk에 의하여 만들어 진 분할창문

통보문들은 절대로 함께 묶어 져서 전송되지 않는다. 이것은 통보문들을 보내고 접수하는 관계를 더 쉽게 구별하게 만든다. 대화는 실시간적이다. 즉 통보문은 타자된것이 있으면 일제히 모든 수신자의 화면에 펼쳐 진다. 이런 방법으로 대화는 두 사람이라든가 한사람이 완료할 결심을 가질 때까지 계속된다. talk는 두 사용자중의 한사람이 새치기건을 누름으로써 완료된다.

만일 자기 체계에 ytalk지령을 가지고 있다면 여러명의 사용자와 인터네트형식의 담화를 진행하기 위하여 이 지령을 리용할수 있다. talk는 두명에게만 제한된다.

11.3 대화의지(mesg)

통신(한 방향 또는 쌍방향)은 말단상에서 어떤 중요한 프로그람의 출력을 다른 사람이 볼수 있는 불리한 측면을 가지고 있다. 다른 사람의 말단에 《쓰기》할수 있는가 그와 《대화》할수 있는가 하는것은 mesg설정에도 의존된다.

```
mesg n
```

이 지령은 다른 사람이 말단에 쓰기하는것을 막는다. y는 이러한 통보문접수를 가능하게 한다. 말단기의 상태를 알려면 간단히 인수없이 mesg를 리용하시오.

```
$ mesg
is y                통보문들을 접수할수 있다
```

mesg는 사용자가 가입할 때마다 쉘이 실행시키는 시동파일에 보존되는 하나의 지령이다. 이 파일의 내용은 말단이 그 사용자가 요구하는 상태로 설정되게 해준다. 후에 이 파일과 그 변수들에 대해서 론의할것이다. 만일 쓰기도 대화도 하지 않는다면 mesg설정을 검사하여야 한다.

11.4 사용자들에 대한 세부자료(finger)

finger(버클리에서 개발된)는 사용자들에 대한 세부정보를 보여 주는 지령이다. talk와 마찬가지로 이 지령은 국부체계와 원격체계에 다 리용될수 있다. 기정적으로 finger는 단순히 등록된 모든 사용자들의 목록을 국부기계에 생성한다. 그러나 원격사용자들에 대한 세부를 얻으려면 앞붙이 @로 된 주콤퓨터이름과 함께 리용할수 있다.

```
$ finger @jupiter
Login     Name               Tty    Idle     Login   Time   Where
Henry     henry blofeld      *p1               Sep   4 12:15
Sumit     sumitabha  das       2               Sep   4 08:27
Root      root                 4      45       Sep   4 10:19
Charlie   charlie greene      p0               Sep   4 12:14
```

두번째 마당은 /etc/passwd로부터 얻은 사용자의 이름을 보여 준다. 세번째 마당은 사용자의 말단을 보여 준다. 여기서 별표는 쓰기허가를 가지지 못하였다는 뜻을 가진다. 네번째 마당은 말단에 마지막 건이 입력된 때로부터 지나간 시간을 보여 준다. 여기서 root는 45분동안 놀고 있다. 마지막마당은 대체로 빈 공백이지만 사무실위치(/etc/passwd의 다섯번째 마당으로부터 주어 진다.)나 주콤퓨터이름을 보여 줄수 있다.

who와는 달리 finger는 원격기계에 한명의 사용자에 대한 세부를 제공해 줄수도 있다.

```
$ finger romeo@jupiter
[jupiter]
Welcome to Linux version 2.2.5 at jupiter!
12:22pm  up  3:51,  2 users,  load average: 0.00, 0.03, 0.00

Login: romeo                      Name: romeo cox
Directory: /home/romeo            Shell: /bin/bash
On since Sat Sep 4 12:14 (EST) on tty2, idle 0:01 (messages off)
Mail forwarded to romeo@yahoo.com
New mail received Sat Sep 4 16:45 1999 (EST)
Mail last read Sat Sep 4 11:28 1999(EST)
No Plan.
```

여기서 새로운것을 볼수 있다. 조작체계는 Linux인데 romeo는 bash쉘을 리용하고 있고 말단을 비능동화하였으며(message off) 아무런 계획도 가지고 있지 않다(no plan). 만일 romeo가 가입되여 있지 않으면 출력은 마지막가입시간을 보여 줄것이다. romeo는 또한 자기의 .forward파일(13.5)안에 어떤 항목을 넣음으로써 romeo@ yahoo.com에 자기의 우편물을 보냈다. finger는 여기서 사용자가 마지막으로 접수하고 우편을 읽은 시간과 날자를 보여 준다.

만일 사용자의 등록가입이름을 모른다 해도 부분적으로나마 알고 있으므로 finger의 인수로서 첫 이름 혹은 마지막이름을 리용할수 있다. 만일 henry가 /etc/passed에

```
hanry:x:200:50:henry james:/home/henry:bin/ksh
```

의 내용으로 등록되였다면 사용자는 그의 마지막이름도 인수로 하여 finger를 리용할수 있다.

```
finger james
```

참고

만일 어떤 사람의 전화번호나 주소를 찾으려면 그 사람의 전자우편주소와 함께 finger를 리용한다. 만일 전자우편주소도 모르면 그의 첫번째 혹은 마지막이름을 리용하시오. 아예 모르면 운수가 좋아야 정보를 얻을수 있다.

plan과 project파일

특별히 휴가를 받으려면 일정표와 다른 중요한 정보들은 남겨 두고 떠나야 한다. 모든 사용자들에게 우편을 보낼수는 없으므로 finger를 리용하여 $HOME/.plan과 $HOME/.project파일의 내용을 현시할수 있다.만일 romeo가 자기의 홈등록부에 이 2개의 파일을 가지고 있다면 finger는 표준출력의 마지막에 다음의것을 보통출력으로 보여 준다.

```
Project:
The tulec2 project should be completed by Feb 28, 2001
Plan:
The DTP people have to be contacted on Feb 25, 2000
A number of diagrams need to be drawn with illustrator.
```

주해

대부분의 사람들은 finger가 사적비밀을 로출시킨다고 꺼려하고 있다. 그들은 본인의 승인없이 개인자료가 다른 사람들에게 공개되지 말아야 한다고 주장한다. 그리하여 finger봉사는 많은 체계에서, 인터네트상에서 무시 당하고 있다.

다음절들에서는 TCP/IP가 발생한 초기부터 발전해 온 도구들을 론의할것이다. 모든 UNIX체계들은 이 도구들을 제공하며 그것들중의 대부분이 아직도 인터네트상에서 리용되고 있다.

11.5 원격가입(telnet)

초기에 TCP/IP응용프로그람들의 발전은 기본적으로 두가지 원천 즉 DARPA(인터네트를 창시한 최종적인 단체)와 버클리로 국한되였다. 오늘날에 사람들은 그것들을 DARPA모임이라고 부르며 그의 도구들이 모두 r로 시작된다는 의미에서 **r-편의프로그람**(r-utility)이라고 부른다.

모든 UNIX제공자들은 원격다중사용자체계에 가입하기 위하여 TCP/IP묶음에 DARPA의 telnet와 Berkeley의 rlogin지령을 제공한다.

rlogin은 UNIX기계도 쓸수 있지만 telnet는 그렇지 못하다. telnet에서는 봉사기쪽이 IBM, 디지털, 싼 등이 제공하는 시분할체계(time-sharing system)이여도 된다. 만일 국부망(혹은 인터네트)안에 있는 어떤 주콤퓨터에 대한 등록자리를 가지고 있다면 인수로서 주콤퓨터이름 혹은 IP주소와 함께 이 도구를 리용할수 있다.

```
$ telnet saturn
Trying 192,168.0.1              /etc/hosts에 의해 정해 진 이름
Connected to saturn
Escape character is '^]'
```

```
login: henry              사용자이름과 통과암호입력
Password:  ********
Last login: Thu Sep 23 12:19:42 on ttyp0 from ppp112-202.pppc.
You have mail.
$ _
```

프롬프트상태에서 exit지령을 리용함으로써 telnet를 완료할수 있다. 일단 프롬프트가 나타나 사용자가 타자한것은 원격기계에 보내지며 그의 콤퓨터는 어떤 다른 벙어리말단(dumb terminal)처럼 동작한다. 사용자는 원격기계가 리해할수 있는 지령을 보낼수 있으며 거기서 그것들을 실행할수도 있다. 그리고 말단에 현시된 출력은 계속 남아 있다. 원격기계의 조작체계판본을 찾아 보자.

```
$ uname -r
5.7                SunOS 5.7를 가진 Solaris 7
```

두 기계가 UNIX의 같은 판본을 리용할 때는 때때로 자기가 어느 기계에 등록되였는지 찾기가 어렵다. bash쉘에서는 프롬프트에 주콤퓨터이름을 쓰던것(17.3)대신에 telnet를 리용하게 된다.

참고

때때로 자기가 국부적으로 리용하던 건들로 문자들을 지울수 없는 경우가 있다. 이것은 원격기계가 이 목적을 위하여 서로 다른 건을 할당 받았다는것을 의미한다. 그 경우에 [Delete], [Ctrl-h] 혹은 [Backspace]를 쓴다. 그래도 되지 않으면 19.1과 3.5를 보시오.

11.5.1 telnet>프롬프트

telnet가 주소없이 리용되면 체계는 내부지령을 리용할수 있는 곳에 telnet>프롬프트를 현시한다. 그러면 telnet의 내부지령의 하나인 open으로 등록대화(login session)를 호출할수 있다. 즉

```
telnet> open 202.54.54.35              이때 IP주소를 리용한다
Trying 202.54.54.35...
Connected to 202.54.54.35
Escape character is '^]'.
      ......
```

telnet를 쓸 때 자기 기계에 상주하는 파일의 이름을 검사해야 할 필요가 있을수 있다. 《탈퇴문자》(escape character)는 사용자가 국부기계에서 어떤 지령을 실행시킬수 있도록 telnet>프롬프트상으로 일시 탈퇴하게 해준다. 이것을 실현하기 위하여 [Ctrl-]](Ctrl건과]건)을 누르시오. 그다음 국부기계에 파일들을 기입하기 위하여 UNIX의 지령 ls와 함께 !를 리용할수 있다.

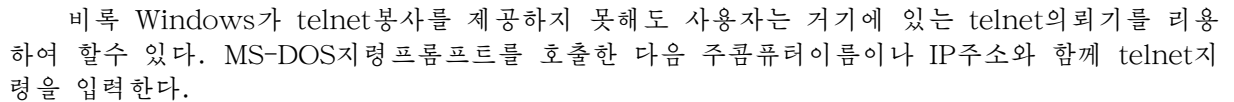
```
$ [Ctrl-]]
telnet> !ls -1 *.sam            국부기계상에서 실행
```

이 프롬프트로부터 자체로 자기의 현재대화나 telnet를 완료할수 있다. 이때는 다음의 두가지 지령중의 하나를 사용하시오.

- logout로 등록에서 탈퇴한다. 이 지령은 물론 telnet도 완료한다.
- close로 현재접속을 끝낸다. 만일 인수로서 주콤퓨터이름과 함께 telnet를 호출하였다면 이 지령도 logout와 같이 동작한다. 그렇지 않으면 단지 현재접속만을 끝내며 그때에는 여기서 open지령으로 새로운 접속을 열수 있다. 그러나 일부 체계들은 다르게 동작한다.

주해

비록 Windows가 telnet봉사를 제공하지 못해도 사용자는 거기에 있는 telnet의뢰기를 리용하여 할수 있다. MS-DOS지령프롬프트를 호출한 다음 주콤퓨터이름이나 IP주소와 함께 telnet지령을 입력한다.

11.5.2 열람기에서의 telnet

Web를 탐색하면서 고속telnet접속이 요구되는 경우에는 Netscape열람기를 리용할수 있다. 그러자면 그 열람기의 꼭대기에 있는 작은 창문에 다음의 문자렬을 입력해야 한다.

```
telnet://sasolution.com[Enter]                이것은 URL이다
```

먼저 통신규약(telnet)을 입력하고 그다음 :과 사선(/) 두개가 뒤따른다. 다음 Enter건을 입력하기전에 그 싸이트의 FQDN을 입력한다. 열람기는 여기서 VT100말단기처럼 동작하는 xterm창문(12.9)을 호출한다. 다음 등록가입하고 자기 작업을 하다가 창문을 닫기전에 exit로 접속을 끝낸다. 그러면 열람기로 되돌아 온다.

telnet는 원격기계를 관리하기 위한 리상적인 도구이다. 대부분의 UNIX프로그람들은 아직까지 이 지령행대면부를 리용하는데 관리에는 흔히 간단한 본문파일편집이 요구된다. 이것은 telnet로 쉽게 된다. 지어는 telnet로 원격기계를 완료할수도 있다.

주해

열람기창문에 입력하는것은 telnet싸이트의 URL(Uniform Resource Locator)이다. 이것은 제14장에서 충분히 설명되는 Web전문용어이다.

참고

주콤퓨터에서 어떤 봉사프로그람이 실행되고 있는가를 알려면 telnet의 두번째 인수로 포구번호를 리용할수 있다. 실례로 host cal.vsnl.net.in 에서 POP3봉사프로그람이 실행되고 있는가를 검사하기 위하여 telnet의 추가적인 인수로서 110(/etc/services에서 POP-3에 리용되는 포구번호)을 리용한다. 즉

```
$ telnet cal.vsnl.net.in 110
Trying 202.54.9.25...
Conneted to cal.vsnl.net.in.
Escape character is '^]'.
+Ok QUALCOMM Pop sever derived from UCB(version 2.1.4
-R3) at cal.vsnl.net.in starting.
quit                                     Quits telnet when in this mode
+Ok Pop server at cal.vsnl.net.in signing off.
```

그러면 POP3봉사프로그람이 실행되고 봉사기로부터 보내는 POP와 관련된 통보문들을 나타난다. 여기서 telnet를 완료하려면 POP3의 quit지령을 리용해야 한다.

11.6 통과암호를 리용하지 않는 원격가입(rlogin)

rlogin은 쉽게 원격가입하게 하는 버클리의 도구인데 오직 UNIX체계에서만 작용한다. rlogin은 사용자이름이나 통과암호가 없이도 자기의 동일한 원격등록자리로 가입시켜 준다.

```
$ rlogin jupiter
Last login: Mon Oct 25 09 :26:46 from saturn
$ _                     사용자이름이나 통과암호를 입력하지 않고 가입하였다
```

여기서 사용자는 자기 인증이 없이 원격체계상에서 자기의 등록자리에 접근하였다. 두 기계는 그들

의 /etc/passwd파일들에 이 사용자에 대한 등록항목을 가지고 있어야 한다. 물론 그것만으로는 부족하다. 즉 원격기계는 이것이 가능하도록 적당히 구성되여야 한다. 11.10에서 그 구성에 대해 론의한다.

-l선택항목을 리용하여 국부기계에 류사한 등록자리가 없어도 다른 사용자등록자리로 가입할수 있다.

```
rlogin -1 franklin saturn              사용자는 사용자이름 frankin으로 가입한다
```

이때 사용자는 통과암호를 입력해야 한다. 그리고 국부기계에 frankin사용자등록자리를 가지고 있을 필요는 없다. 그러나 《적당히》 구성하면 암호를 리용하지 않고도 사용자입장을 허락한다.

rlogin은 등록가입대화가 완료되던 방식으로 완료된다. 즉 [Ctrl-d], exit 혹은 logout지령으로 완료된다.

참고

Linux기계에서 같은 등록자리로 UNIX기계에 접근하기 위하여 rlogin을 리용하고 있는 경우 -k선택항목으로 커베로스인증(Kerberos authentication)을 없애지 않으면(Linux기계에서) 암호가 있어야 접근할수 있다.

11.7 파일전송규약(ftp)

TCP/IP는 파일을 전송하기 위한 두가지 지령 즉 DARPA의 ftp(file transfer protocol:파일전송규약)와 버클리의 rcp를 제공한다. 이 장에서는 rcp보다 널리 리용되는 ftp를 론의한다. 여기서 telnet에서 리용하던 그 주콤퓨터상에서 ftp를 리용한다.

```
$ ftp saturn
Connected to saturn.
220 saturn FTP server (Version wu-2.4.2-academ[BETA-17](1) Tue Jun 9 10:
43:14 EDT 1998) ready.
Name (saturn:sumit): henry
331 Password required for henry.
Password: *********
230 User henry logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> _
```

한대의 주콤퓨터가 여러개의 봉사를 제공한다(그것이 전송량을 처리할수 있는 경우). 여기에는 같은 가입방식이 적용되며 사용자는 홈등록부안에 있게 된다. ftp를 완료하려면 close지령을 주고 그다음 bye나 quit지령을 리용할수 있다.

```
ftp> close              필요에 따라 이것을 뛰여 넘을수 있다
221 Goodbye.
ftp> bye                quit를 리용할수도 있다
```

ftp는 ftp>프롬프트상에서 리용할수 있는 내부지령을 가지고 있으며(표 11-1) 그 지령들중의 일부는 UNIX에서와 같은 이름을 가지고 있다. 실례로 등록부를 만들고 제거할수 있으며(mkdir와 rmdir) 파일속성을 변화시킬수 있다(chmod).

표 11-1. ftp지령들

지 령	의 미
open hname	주콤퓨터 hname에 접속한다
user uname	사용자이름 uname으로 접속한다
bye 혹은 quit	ftp를 완료한다
mkdir, rmdir,cd,pwd	지령들이 원격기계에서 실행되는것을 제외하고는 유닉스에서와 같다
cdup	원격기계의 어미등록부에로 이동한다
ls	원격기계상에서 파일들을 렬거한다
!ls	국부기계상에서 파일들을 렬거한다
lcd ..	국부기계상에서 어미등록부에로 이동한다
get f1 f2	원격지의 파일 f1을 국부기계의 파일 f2로 복사한다(f2가 지적되지 않았다면 파일이름은 같다)
restart n	n개의 바이트를 뛰여 넘은 다음 중단되였던 원격기계로부터 파일 전송을 다시 시작한다
reget flname	중단되였던 원격기계로부터의 flname파일의 전송을 다시 시작한다(Linux에서 리용)
put f1 f2	국부기계의 파일 f1을 원격기계의 파일 f2로 복사한다(f2를 지적하지 않으면 이름은 같다)
mget filenames	여러개의 원격파일들을 국부기계에 복사한다(통용기호들을 원격기계에서 번역)
mput filenames	여러개의 국부파일들을 원격기계에 복사한다(통용기호들을 국부기계에서 번역)
binary 혹은 ascii	2진방식 혹은 ASCII방식으로 전송형태를 설정한다
delete와 mddelete	원격기계상에서 한개 또는 여러개 파일을 지운다
Verbose	반전스위치: 통보문을 개통/차단시킨다
Prompt	mput와 mget를 리용할 때의 호상작용동작을 개통/차단시킨다

사용자는 파일체계를 조종하기 위하여 ftp의 pwd, cd 그리고 ls지령묶음을 리용할수도 있으며 UNIX체계에서 리용하던 방식대로 파일들을 렬거할수도 있다.

```
ftp> pwd
257 "/home/henry" is current directory.
ftp> cd download
250 CWD command successful.
ftp> ls                  어느 ls를 리용하는가?
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.              UNIX지령
total 142720
drwxr-xr-x   2 root      users         1024 Sep  7 15:34  .
drwxr-xr-x   8 adeb      users         1024 Jun  2 11:26  ..
-rw-r--r--   1 root      root         30520 Feb 17 1999 1req.htm
-rw-r--r--   1 root      root         48761 Feb 17 1999 2pre.htm
-rw-r--r--   1 root      root         37899 Feb 17 1999 3inst.htm
...
```

비록 ftp의 내부지령들이 존재한다 해도 ls지령은 /bin등록부안에 있는 조작체계의 ls지령을 리용한다. 그것은 ls지령의 출력이 조작체계가 수행하는 방식에 따라 기계마다 다르기 때문이다.

국부기계지령들은 앞불이 !를 요구한다. !는 cd지령과 함께 동작하지 못하므로 ftp는 그 일감을 수행하기 위하여 lcd(local cd)지령을 제공한다. 사용자가 어미등록부로 옮겨 가고 싶으면 cd..(원격)과 lcd..(국부)지령을 리용할수 있다.

11.7.1 파일들의 전송

파일들은 전송목적을 위하여 두가지 형태 즉 ASCII(본문)와 2진형태를 취한다. 모든 실행파일, 도형파일, 문서파일, 다매체파일들은 2진형태에 속한다.

이 파일들에 대해서는 전송하기전에 binary로 전송방식을 설정해야 한다.

올리적재(put와 mput)

만일 독자가 Web싸이트개발자이라면 자기의 Web싸이트에 Web페지와 도형파일들을 올리적재(upload)해야 할것이다.

put지령은 원격기계에 penguin.gif파일을 보낸다.

```
ftp> binary
200 Type set to I .
ftp> put penguin.gif              같은 이름으로 복사
local: penguin.gif remote: penguin.gif
200 PORT command successful.
150 Opening BINARY mode data connection for penguin.gif.
226 Transfer complete.
6152 bytes sent in 0.04 seconds (150.20 kbytes/s)
```

목적파일이름을 변화시킬수도 있으며 mput로 여러개의 파일들을 복사할수 있다.

```
put penguin.gif pelican.gif
mput t*.sql              *는 국부기계상에서 해석된다
```

내리적재(get와 mget)

원격기계로부터 파일들을 내리적재(download)하려면 get와 mget지령이 필요하다. 이때 verbose지령으로 모든 통보문들을 차단시킬수 있다.

```
ftp> verbose              불필요한것들을 다 차단시킨다
ftp> ls
drwxr-xr-x   14  888       999    4096  Jun  15  16:46  communicator
drwxr-xr-x    2  888       999      26  May  14  00:47  communicator_for_france
-rw-r--r--    1  888       999  323393  Sep   7  17:22  ls-lR
-rw-r--r--    1  888       999   28360  Sep   7  17:22  ls-lR.gz
    ......
ftp> binary              대부분의 체계에서 기정이다
ftp> get ls-lR.gz
ftp>
```

put지령에서와 같이 목적파일이름을 변화시킬수 있으며 mget지령으로 여러개의 파일들을 복사할수도 있다.

```
get ls-lr.gz netscape_filelist
mget t*.sql            *는 원격기계상에서 해석된다
```

인터네트상에는 체험판프로그람과 공개령역쏘프트웨어를 내리적재하도록 해주는 여러개의 싸이트들이 있다. 거기서는 매 사용자에 대한 등록자리와 통과암호를 받지 못한다. 이 싸이트들은 특별한 등록자리 즉《닉명》이라는것을 제공하는데 사용자들은 이 등록자리로 싸이트들에 가입하게 된다. 비록 의무적이지는 않지만 사람들은 자기의 전자우편주소를 통과암호로 제공하기도 한다. 이 싸이트들을 **닉명 ftp**(anonymous ftp)싸이트라고 한다. 우리는 오직 닉명ftp싸이트들로부터 파일을 내리적재할수 있다.

참고

기정적인 파일전송형태를 알려면 ftp>프롬프트상에서 type지령을 리용하시오. 일반적으로 2진방식(binary)을 취한다. 2진방식에서의 작업은 ASCII파일들도 충분히 전송될수 있기때문에 유리하다.

참고

mput와 mget는 일반적으로 전송되는 모든 파일에 대하여 응답을 요구하는데 이것은 더 빠른 전송을 위하여 비대화적인 동작이 필요할 때에는 골치거리로 될수 있다. 대화동작을 없애려면 prompt지령을 리용하시오. 이 지령은 반전스위치처럼 동작한다. 즉 대화동작을 능동으로 하려면 다시 그 지령을 리용할수 있다.

11.7.2 ftp>프롬프트로부터의 접속

ftp는 아무런 인수가 없이 리용되면 ftp>프롬프트를 현시한다.
그다음 open지령으로 접속할수 있다.

```
$ ftp
ftp> open saturn
Connected to saturn.
220 saturn FTP server (Version 2.1WU(1)) ready.
```

일반적으로 가입할수는 있지만 사용자세부를 정확치 못하게 입력하였다면 ftp는 그사람의 입장항목을 거절한다. 그러나 그 싸이트에로의 접속은 유지한다. 그 싸이트에 가입하려면 user지령을 리용해야 하며 그다음 보통가입순서를 통해서 가입한다.

```
ftp> user charlie
331 Password required for charlie.
Password: **********
```

11.7.3 ftp를 비대화식으로 리용하기(.netrc)

사용자들은 언제나 닉명ftp싸이트에 꼭 같은 사용자이름과 통과암호를 리용한다. 그러면 왜 매번 그것들을 입력하며 파일로 그것들을 보관하지 않는가? ftp는 그렇게 하도록 한다. ftp는 또한 매 주콤퓨터에 대해서 개별적으로 인증파라테터들을 지적하도록 하게 해준다. ftp가 그것들을 보관하기 위하여 리용하는 파일은 $HOME/.netrc이다.

.netrc파일안에는 3개의 내용이 있다. 2개는 주콤퓨터 jupiter와 neptunc에 대한것이다. 다른것은 꼭 같은 사용자이름과 통과암호를 리용하는 모든 닉명ftp싸이트들을 위한 항목이다. 이 파일에서 매행은

ftp싸이트에 접속하기 위한 파라메터를 지적한다.

```
machine jupiter login sasol password l1u2dw3ig
machine neptune login romeo password ble6e37nn
default login anonymous password romeo@vsnl.com
```

첫번째와 두번째 행은 사용자가 개별적인 등록자리를 가지고 있는 "개별적"ftp싸이트에 적용된다. machine, login, password는 열쇠단어이며 매 열쇠단어뒤에 대응하는 파라메터가 따른다. 마지막행은 열쇠단어 default를 포함하고 있으며 일반적으로 다른 모든 싸이트들에 적용된다. 여기서 이 행은 닉명ftp싸이트에 리용된다. 이 행은 이전 행보다 한개 단어만큼 적게 파라메터를 가진다.

이 파일은 정확한 본문형식의 통과암호를 가지고 있으므로 어떤 그룹이나 다른 사람들에게 chmod 600.netrc로 읽어 지지 않도록 만들어 져야 한다. ftp의뢰기는 프롬프트를 내보내지 않고 jupiter, neptune 그리고 닉명ftp싸이트에 가입하기 위하여 이 파일을 리용할것이다.

자동파일전송

대화없이 파일을 전송하는것이 어떤가? ftp는 표준입력으로부터 지령들을 받아 들이므로 이 모든 지령들을 파일안에 보관할수 있다.

```
$ cat ftpparam.lst
cd download
prompt              비대화적인 전송
mget *.gif
bye
```

이것은 jupiter싸이트의 download등록부로부터 GIF파일들을 비대화적으로 얻는데 리용하여야 할 4개의 ftp지령들을 보여 준다. 이전에 .netrc파일에서 보여 준것처럼 가입하기 위하여 같은 파라메터들을 리용한다.

다음과 같이 ftp의 표준입력을 다시 고쳐 써야 한다.

```
$ ftp jupiter < ftpparam.lst
$ _
```

지령은 jupiter안의 sasol등록자리로부터 파일들을 복사한후 프롬프트에 돌려 준다.

11.7.4 ftp의 기타 기능

중단된 파일전송을 재전송하기

이것은 흔히 파일전송이 중단되는 인터네트상에서 제기되는 문제이다. 일부 ftp판본들은 중단된 점으로부터 전송을 다시 시작하게 해준다. get를 리용하기전에 restart지령을 리용해야 한다.

```
restart 846913          뛰여 넘어야 할 바이트수
get exceptions          846913byte를 뛰여 넘은후 파일전송
```

Linux

Linux에서 중단된 파일의 재전송은 아주 쉽다. ftp의 reget지령을 리용하시오.

```
reget exceptions          모든 계산을 자동적으로 한다
```

내리적재하지 않고 문서를 보기

문서들을 내리적재하기전에 그것을 보고 싶은 경우가 있다. 그때 대체로 페지화프로그람을 리용할수도 있다.

```
get root-servers.txt /dev/tty             /dev/tty는 사용자의 말단이다
get root-servers.txt - \| more             때때로 등록한다
```

만일 두번째 지령이 사용자의 기계에서 동작한다면 문서를 자동적으로 내리적재하지 않고도 그것을 한 페지씩 전진시키거나 후진시킬수 있다. 그러나 이 기능은 대부분의 체계들에서는 쓰이지 않으므로 자기의 기계에서 동작하지 않는다고 하여 실망하지 마시오.

주해

비록 Windows가 ftp봉사를 제공하지 못해도 ftp싸이트에 접속을 하기 위하여 거기서 ftp의뢰기를 리용할수 있다. MS-DOS프롬프트를 호출하여 여기서 리용된 방법으로 ftp지령을 입력한다.

11.7.5 Web열람기에서의 ftp

한개 파일을 얻으려고 하는 경우 그 파일이 있는 곳을 알려면 열람기로부터 접속하는것이 가장 좋은 방법이다. 모든 열람기들은 FTP규약을 지원하는데 Netscape열람기창문에서 그것을 지적할수 있다.

```
ftp://ftp.javasoft.com[Enter]
```

타자는 이것으로 끝나며 Netscape가 자동적으로 단어 《anonymous》와 통과암호를 제공한다. 우리가 보통 Web페지를 보던 령역은 지금 싸이트의 뿌리등록부구조를 보여 준다(그림11-2).

```
URL창문

Netscape: Directory of /docs
File  Edit  View  Go  Communicator                                         Help
Back  Forward  Reload  Home  Search  Netscape  Print  Security  Stop
Bookmarks  Location: ftp://ftp.javasoft.com/docs/                 What's Related

Current directory is /docs

Up to higher level directory
  100percent/                    Mon Oct  5 00:00:00 1998 Directory
  FirstJolt.fm          136 Kb   Mon Oct  5 00:00:00 1998
  Java.pdf              357 Kb   Mon Oct  5 00:00:00 1998 Portable Document
  LangConstructs.fm     156 Kb   Mon Oct  5 00:00:00 1998
  WhitePaper.pdf.Z      353 Kb   Mon Oct  5 00:00:00 1998 Portable Document
  WhitePaper.pdf.gz     232 Kb   Mon Oct  5 00:00:00 1998 Portable Document
  WhitePaper.pdf.zip    232 Kb   Mon Oct  5 00:00:00 1998 Zip Compressed Dat
  applets/                       Mon Oct  5 00:00:00 1998 Directory
  beans/                         Mon Oct  5 00:00:00 1998 Directory
  books/                         Mon Oct  5 00:00:00 1998 Directory
  codeconv/                      Mon Oct 18 00:00:00 1999 Directory
  cyber2.ppt            581 Kb   Mon Oct  5 00:00:00 1998 Microsoft PowerPoi
  design.pdfs.tar      4927 Kb   Mon Oct  5 00:00:00 1998 Unix Tape Archive
  design.pdfs.zip      2021 Kb   Mon Oct  5 00:00:00 1998 Zip Compressed Dat
  ejb/                           Mon Oct  5 00:00:00 1998 Directory
  industry.pdfs.tar    5364 Kb   Mon Oct  5 00:00:00 1998 Unix Tape Archive
  industry.pdfs.zip    3635 Kb   Mon Oct  5 00:00:00 1998 Zip Compressed Dat
  j2se1.3/                       Tue Feb  1 06:54:00 2000 Directory
  java-media/                    Wed Jan 27 00:00:00 1999 Directory
```

그림 11-2. 열람기에서의 ftp

일단 이 방식을 설정하기만 하면 항행과 내리적재가 쉬워 진다. 밑선을 친 등록부를 찰칵하면 그안의 파일이 렬거된다. 거기서 어떤 파일을 찰칵하면 다음의것들중에서 하나가 동작한다.

- 2진파일이라면 그 파일이 디스크에 보관되기전에 목적파일이름을 요구하게 된다.
- 본문파일이라면 화면에 파일의 내용을 보여 준다. 그 파일을 보관하려면 File>Save As의 Netscape차림표를 리용해야 한다.

보통 마우스조작으로 전반적인 일을 수행할수 있지만 다음의 방법은 제한된다는것을 알아야 한다.

- 한번에 오직 한개 파일만 내리적재할수 있다.
- 봉사기에 올리적재를 할수 없다.
- 전송이 그 사용자의 직접적인 조작이 아니라 외적인 현상에 의해 중단되면 다시 전송될수 없다.

ftp의 기능들을 리용하기전에 그것이 가능한가 확인하여야 한다. 열람기구성창문(Edit> Preferences>Advanced)은 전자우편주소를 닉명ftp통과암호로 설정하는 검사칸을 찰칵하게 해준다.

참고

만일 Java문서를 가지고 있는 파일의 완전한 경로를 알고 있다면 파일경로이름과 함께 URL 문자렬로 그 파일을 직접 검색할수 있다.

ftp://ftp.javasoft.com/docs/tutorial.zip

11.8 원격파일복사(rcp)

rcp는 ftp지령과 비슷한 버클리의 지령이다. 그것은 원격체계에서부터 원격체계에로 파일들과 등록부들을 복사하는데 리용되는것을 제외하고는 cp지령처럼 동작한다. 원격기계에 있는 파일은 :으로 분리된 주콤퓨터이름과 파일이름의 조합으로 rcp에 지적된다. 그러면 host uranus로부터 파일을 복사하는 두가지 방법을 보자.

```
rcp uranus: /home/henry/cent2fah.pl c2f.p1
rcp uranus:/home/henry/cent2fah.pl .              같은 이름으로 복사
```

만일 henry의 홈등록부로부터 파일이 복사되면 지령행이 좀 더 짧을수 있다.

```
rcp henry@uranus:cent2fah.pl c2f.pl
rcp nenry@uranus:cent2fah.p1 .
```

여러개의 파일을 복사하려면 마지막인수가 등록부이여야 한다. 쉘통용기호들도 사용할수 있다. 그러나 통용기호들이 원격기계에서 해석될것인가 국부기계에서 해석될것인가는 그것들이 인용부호화되였는가 안되였는가에 관계된다.

```
rcp uranus:/home/henry/"*" .              *는 원격기계에서 해석된다
rcp * uranus:/home/henry                  *는 국부기계에서 해석된다
```

rcp가 실지로 ftp에 의해 교체된다 하여도 거기에는 ftp보다 우월한 한가지 중요한 기능이 있다.즉 그것은 보조등록부를 복사할수 있다. 그 기능은 -r선택항목으로 실현할수 있다.

```
rcp -r uranus:/home/henry/cgi-bin.              cgi-bin 등록부나무를 복사
```

rcp는 UNIX계에서 만들어 졌으므로 제공되는 원천과 목적파일이름을 다 요구한다. ftp와는 달리 전송처리과정을 보여 주지 않는다.

참고

rcp를 리용할 때 사용자이름@주콤퓨터이름의 형식을 리용해야 한다. 만일 지령 rcp foo juliet@uranus:docs를 리용한다면 juliet의 홈등록부의 절대경로가 무엇이든 관계 없다. 그 파일은 docs등록부에도 복사될것이다.

주해

ftp가 비록 재귀적인 복사를 못한다 해도 보존파일(archive)을 복사할 때 리용할수 있다. 이 파일은 한개의 파일로 결합된 파일들의 묶음이다. tar지령(22.9)은 전체 등록부나무를 한개의 파일로 묶어 놓을수 있다. 이 보존파일은 compress 혹은 gzip(6.19)로 압축될수 있으며 그러면 ftp로 혹은 전자우편첨부물(13.7.6)로 어떤 사람에게 보낼수 있다.

11.9 원격지령실행(rsh)

가입절차를 통하여 가입하지 않고 CPU에 의존하는 프로그람을 실행시키기 위해 보다 큰 체계의 자원이 요구되는 경우가 있다. rsh는 사용자가 원격기계상에 있는 어떤 프로그람을 실행시키기 위하여 그 콤퓨터에 요청할수 있게 해준다.

이를 위해서는 그 기계이름과 지령이 필요된다.

```
rsh uranus ls -l                사용자의 홈등록부에서 실행된다
```

ls가 원격기계에서 실행되여도 출력은 자기 말단에 현시된다. 출력을 그 원격기계에 파일로 보관하고 싶다면 ">"문자앞에 "\"을 주어야 한다.

```
rsh uranus ls -l \> dir.lst              >는 원격기계에서 해석된다
```

"\"이 빠지면 출력을 국부파일에 보관한다. 만일 원격기계에서 해석되는 통용기호를 리용한다면 그것들은 인용부호화하든가 앞에 "\" 을 주어야 한다. rsh는 또한 henry가 frankline등록자리로 같은 지령을 원격실행하게 해주는 -l선택항목을 가지고 있다.

11.10 r-편의프로그람에 대한 보안시행

버클리는 원격기계에 더 쉽게 접근하게 해주는 r-편의프로그람들을 만들었다. 만일 rlogin, rcp, rsh를 고유한 의미대로 실행시키지 못한 경우 이 절은 권한 없는 사용자들까지도 무조건 읽어야 한다.

UNIX체계는 구성파일들에 어떤 내용에도 따르지 않는다. 그러므로 r-편의프로그람들을 리용하기전에 그것들을 삽입해야 한다. UNIX는 두가지 준위의 인증을 제공한다.

- 주콤퓨터의 모든 사용자들에게 적용되는 체계준위. 여기서 인증은 원격기계에 있는 /etc/hosts.equiv파일에 의해 조종된다.
- 어떤 사용자가 $HOME/.rhosts파일안에 자기의 등록자리에 대한 접근제한을 설정할수 있는 사용자준위.

이 파일들의 내용은 다음의 두가지를 결정한다.

- rlogin을 통과암호없이 리용할수 있는가 없는가를 결정한다.
- rcp와 rsh가 통과암호를 요구하지 않으므로 모든 곳에서 이 지령들을 리용할수 있는가 없는가를 결정한다.

체계준위인증을 먼저 보자. 어떤 사용자가 saturn상에서 주콤퓨터 uranus에 접속하는데는 3가지 가

능한 상태가 있다고 가정하자. uranus의 /etc/hosts.equiv파일안에 있는 어떤 행은 다음의 내용을 포함할수 있다.

- **단어 saturn만으로**: 이것은 어떤 사용자가 통과암호를 리용하지 않고 saturn으로부터 uranus안에 있는 자기 등록자리로 가입할수 있다는것을 의미한다. 이때 주콤퓨터는 **믿을수 있다**(trusted)고 하며 사용자는 원격주콤퓨터안에 있는 자기의 등록자리와 같은 **사용자등가성**(user equivaleuce)을 가진다고 한다. 만일 내용이 주콤퓨터이름을 가지고 있다면 같은 등록자리에 대해서는 통과암호가 필요 없다.
- **두개 단어 saturn henry**: 그다음 원격체계는 보다 허가적인 방식으로 동작한다. henry는 통과암호를 리용하지 않고(물론 뿌리는 제외하고) saturn으로부터 uranus안에 있는 어떤 등록자리에 가입할수 있다. 이것은 어떤 등록자리를 리용하려는 henry에게 절대적인 권한을 준다(rlogin에 대해서는 -l선택항목이 필요하다).
- **기입내용이 없이**: henry는 통과암호를 제출한후에야 가입할수 있다.

차단이 없는 접근은 보안에 나쁜 영향을 주므로 체계관리자들은 이런것을 허용하지 않는다. 더 좋기는 사용자준위에서 .rhosts파일과 함께 접근을 시행하는것이다. 이 파일은 관계되는 사용자에 의해 편집된다.

같은 실례를 고찰하자. 그러나 여기서는 uranus의 /etc/hosts .equiv안에 saturn과 관련된 내용항목이 없다고 가정하자. 만일 henry(saturn으로부터)가 통과암호를 리용하지 않고 uranus안에 있는 charlie의 등록자리로 접근하는것이 허락되여 있다면 charley의 .rhosts는 다음과 같은것을 가지고 있어야 한다.

```
saturn henry
```

그러나 그 행이 saturn이라는 단어를 포함하면 같은 사용자만이(즉 charlie) 통과암호없이 가입할수 있다. 이 원리는 .rhosts의 행에 두개의 마당이 있다면 그에 해당한 사용자를 믿고 그렇지 않으면 그와 같은 사용자를 믿는다는것이다.

rcp와 rsh도 실행되는가 안되는가는 제외하고 etc/hosts.equiv와 .rhosts파일을 리용한다.

참고

오히려 사용자준위에서 보안을 시행하는것이 더 낫다. 가장 안전하고 가장 제한적인것으로 되게 하자면 관계되는 사용자의 .rhosts안에 바로 주콤퓨터이름을 기입하게 하는것이다.

11.11 인터네트의 서막

지금까지 UNIX체계의 필수적인 TCP/IP도구들을 리용하는 방법을 배웠다. ftp와 telnet와 같은 지령은 인터네트상에서도 광범히 리용된다. 그리고 인터네트와 WWW의 급속한 발전은 새로운 도구들을 수많이 만들어 냈다. 이 도구들은 새 소식을 보고 우편목록에 가입하여 Web를 열람할수 있게 하였다. Netscap열람기는 또한 리용하기 쉬우며 도형방식인 많은 도구들을 제공한다. 다음장들에서는 TCP/IP를 좀 더 넓은 범위에서 그리고 구체적으로 취급하게 될것이다.

요 약

TCP/IP는 여러가지 조작체계가 실행되는 각이한 종류의 기계에 접속하는데 리용되는 규약의 묶음이다. TCP/IP는 자료가 파케트별로 분해되고 목적지에서 다시 조립되는 파케트절환체계이다. 또한 제공자에 대하여 독립이며 완전한 오유조종기능으로 하여 전송의 믿음성을 담보한다.

TCP/IP는 의뢰기-봉사기모형으로 동작한다. ftp와 같은 의뢰기응용프로그람은 파제를 수행하기 위하여 상대켠 봉사기에 대응하는 프로그람과 통신한다.

봉사기프로그람은 데몬이라고 하는데 배경에서 실행되며 요청을 접수한다. 망에서 모든 주콤퓨터들은 hostname지령의 출력에 나타나는 이름을 가지고 있다. 또한 하나의 주콤퓨터는 점으로 구별되면서 4개의 수들로 구성되는 IP주소를 가지고 있다. TCP/IP는 이름은 물론 IP주소로 주콤퓨터에 접근시키게 해준다. 사용자들이 기억한 이름을 더 쉽게 찾게 해주기 위하여 이름-주소넘기기를 작은 망의 모든 기계들안에 있는 /etc/hosts파일안에 보관한다.

거대한 망들은 주콤퓨터가 완전지정령역이름(FQDN)으로 서술되는 령역이름체계(DNS)를 리용한다. 이 이름은 계층구조로 배렬되면서 점으로 구별되는 이름들의 렬이다.

사용자들은 다른 사용자와 쌍방향으로 대화하려면 mesg를 y로 설정해야 한다. talk는 직결쌍방향대화를 하게 한다. talk는 화면을 두개로 분할하여 전송통보문들을 보기 쉽게 한다. ytalk는 세명이상의 사용자와 통신할수 있게 한다.

finger는 원격기계에 가입된 모든 사용자들의 상세한 정보를 who지령처럼 보여 준다. 또한 가입하지 않은 사용자에 대해서도 많은 상세한 정보를 보여 준다. finger는 요구하는 사용자의 홈등록부안에 있는 2개의 파일 즉 .plan과 .project의 내용을 현시한다.

telnet는 사용자이름과 통과암호로 원격기계에 가입할수 있게 한다. 가입된후 사용자가 만든 파일들은 원격기계에 존재한다. 그러나 지령들은 자기의 국부주콤퓨터에 현시된다. 국부기계에서 지령을 실행시키려면 telnet>프롬프트의 의미를 [ctrl-]]으로 해제해야 하며 !를 지령앞에 붙여야 한다.

rlogin은 원격가입하는데도 리용되는데 같은 등록자리에 대해서는 통과암호를 요구하지 않는다. 다른 등록자리로 접속하려면 -l선택항목과 함께 리용될수 있다.

ftp는 두 주콤퓨터사이에 파일들을 올리적재하고(put와 mput) 내리적재하는데(get와 mget) 리용된다. "닉명"이라는 이름과 전자우편주소가 닉명ftp싸이트에 접속하는데 리용된다. 중단된 파일전송을 다시 할수 있으며 .netrc로부터 인증정보를 읽음으로써 비대화적으로 가입할수 있다.

rcp는 가입하지 않고도 파일을 전송할수 있다. ftp와는 달리 전체 등록부나무도 복사할수 있다.

rsh는 원격기계상에서 지령을 실행하는데 리용된다. 그리고 원격기계와 국부기계상에 지령의 출력을 보관할수 있다.

r-편의프로그람들인 rlogin, rcp, rsh는 오직 적당한 인증이 봉사기쪽에 제공되여야만 리용될수 있다. 체계준위인증은 /etc/host.equiv파일에 의해 조종된다. 인증은 또한 .rhosts에 의해 사용자준위에서 시

행될수 있다.

시험문제

1. 체계관리자가 hostname지령을 리용하는 방법은 몇가지인가?
2. 소케트(soket)란 무엇인가?
3. /etc/hosts에 대한 문제는 무엇인가?
4. FQDN이란 무엇인가?
5. 령역이름 FTP.download.com이 타당한가?
6. 일반적인 망작업데몬들의 이름과 그것들의 기능에 대하여 말하시오.
7. 주콤퓨터 uranus상에서 사용자 brenda와 통신하려면 talk를 어떻게 리용해야 하는가?
8. 다른 사람들이 finger를 리용하여 휴가인지 아닌지를 알게 하려면 어떻게 해야 하는가?
9. 원격기계 saturn에 가입한 모든 사용자들의 상세한 정보를 찾아 보려면 어떻게 해야 하는가?
10. telnet리용중에 telent>프롬프트를 어떻게 호출하는가?
11. 사용자가 국부기계에서 작업하고 있는지 telnet를 리용하여 원격기계에 가입하여 작업하고 있는지 어떻게 확인할수 있는가?
12. 자기의 Web열람기로부터 telnet싸이트 saturn.planets.com에 어떻게 접속할것인가?
13. 통과암호를 리용하지 않고 rlogin을 리용하려면 그 사용자가 해야 하는 기본조건들이 무엇인가?
14. /bin/is의 이름을 변경시킨 경우 ftp의 내부지령 ls를 리용할수 있는가?
15. ftp를 리용할 때 국부기계에 있는 등록부를 어떻게 변화시키는가?
16. 어느 지령으로 닉명ftp싸이트에 파일을 올리적재하는가?
17. 다음의 지령은 무엇을 하는가 ?

```
rsh jupiter date \> .date
```

련습문제

1. TCP/IP망과 전화망과의 차이는 무엇인가?
2. 왜 TCP를 믿음직한 통신규약이라고 하는가?
3. 독립적으로 떨어 져 있는 기계에서 telent를 리용할수 있는가?
4. 주콤퓨터파일이란 무엇이며 그 파일은 무엇을 담고 있는가?
5. 왜 인터네트상에서 주콤퓨터이름이 리용되지 않고 FQDN들만이 리용되는가?
6. finger를 리용하여 어떻게 포구번호를 찾는가?
7. 같은 주콤퓨터상에 있는 두명의 사용자가 공동주콤퓨터로부터 같은 Web페지를 요구한다면 그들이 서로 다른 소케트들을 계속 리용할수 있는가?
8. talk가 통보문 "Your partiy is refusing messages"를 내보낸다면 어떤 일이 있을수 있는가?

9. 어떤 원격기계는 ftp가 아니라 오직 telnet봉사만 제공한다. 원격기계의 /etc/passwd를 자기기계에 보내고 싶은 경우 어떻게 해야 하는가?
10. telnet리용중에 있으면서 국부기계에서 몇개의 프로그람을 실행해야 한다. 국부기계의 쉘프롬프트를 어떻게 호출할수 있는가?
11. telnet로 여러대의 원격기계에 가입해야 한다. 매번 telnet를 호출하지 않고 어떻게 가입할것인가?
12. rlogin이 더 좋은데 인터네트상에서는 왜 일반적으로 telnet가 리용되는가?
13. cp -i지령은 목적파일이 존재할 때만 대화적인 방식으로 동작한다. 만일 목적파일이 존재하지 않는다해도 TCP/IP를 리용해서 자기의 주콤퓨터에 대화적인 방식으로 파일을 어떻게 복사할수 있는가? 이 기능은 어떤 때 동작하지 않는가?
14. ftp내부로부터 국부등록부를 변화시켰을 때 ftp를 완료한후에도 그 변화된 등록부가 그 자리에 있게 된다. 왜 그런가?
15. ftp로 도형파일을 복사하였는데 파일에 오유가 나타난다. 왜 그런가?
16. ftp싸이트로부터 ie4.exe파일이 356789byte만큼 전송되고 중단되였다. 중단된 위치로부터 전송을 다시 시작하려면 어떻게 해야 하는가?
17. 또 다른 주콤퓨터에 있는 자기의 등록자리에로 "ftp"해야 한다. 그러나 주콤퓨터이름과 함께 ftp지령을 리용한다면 다른 등록자리로 가입한다는것을 알게 된다. 그런 일이 생기면 그 상태를 어떻게 수정할것인가?
18. 닉명ftp싸이트에 있는 본문파일을 내리적재하지 않고 어떻게 볼수 있는가?
19. 닉명ftp싸이트에 비대화적으로 어떻게 접속할것인가?
20. 열람기로부터 ftp를 리용하는데서 불리한 점은 무엇인가?
21. ftp가 더 좋은데 어떤 때 rcp를 리용해야 하는가?
22. 만일 주콤퓨터 neptune에 있는 /etc/host .equiv가 단어 mars를 포함하고 있다면 그 내용항목을 어떻게 해석하여야 하는가?
23. 우의 내용을 제거한다면 어떻게 되겠는가?
24. mars상의 사용자 henry가 임의의 등록자리에로 또는 통과암호를 리용하지 않고 오직 자기의 등록자리에로만 접근하도록 되여 있다면 neptune상의 원격기계는 어떻게 구성되여야 하는가?

제 12 장. X Window체계

UNIX체계에서는 지령행과 벙어리말단(dumb terminal)이 사용자대면부로 리용되여 왔다. 그러나 말단들은 과학적이며 공학적인 응용프로그람들의 요구에 맞는 높은 화질의 도형을 생성할수 없다. 애플과 마이크로쏘프트는 통속적인 도형사용자대면부(GUI)를 만들었으며 사용자들은 복잡한 지령행문법에 싫증을 느끼고 있었다. 이것이 바로 매써츄쎄쯔기술연구소(MIT)의 연구사들이 X Window체계를 개발하던 때에 UNIX계에서의 상황이다.

X Window체계는 MIT에서 아테네계획(Project Athena)의 부분으로 개발되였다. X(즉 X Window)는 초기에 어떤 특별한 조작체계를 념두에 두고 설계되지 않았다. 그러나 X는 지금 모든 UNIX제공자들에 의하여 표준적인 창문체계로 쓰이고 있다. 물론 쓸모 있는 Windows판본들도 있다. 1987년에 MIT에서 X의 11판(X11)을 내놓았다. X11은 또 여러개의 발매판(release)을 가지고 있는데 마지막발매판이 6.4(X11R6.4)이다. X는 The Open Group이 조종하고 관리한다.

이 장에서는 X Window체계(X Windows가 아니다.) 즉 X봉사기와 의뢰기(도구)들을 서술한다. 우에서 X의 두가지 측면을 관찰한다. 첫째로 마우스를 항행도구로 리용하여 어떤 GUI를 다루던 방법대로 X를 다루는 법을 배운다. 둘째로 X가 어떻게 설계되였으며 망에서 어떻게 실행되는가를 배운다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- X를 리용하는것이 왜 필요한가를 배운다(12.1).
- X를 기동시키고 X와의 대화를 완료하는 방법을 배운다(12.3).
- 봉사기와 의뢰기의 역할에 대하여 배운다(12.4).
- 원격사용자에 의하여 지령이 xhost와 함께 실행되게 한다(12.5.1).
- 원격기계에 DISPLAY변수와 -display선택항목으로 출력을 현시한다(12.5.2).
- X가 두개의 구성요소로 어떻게 분할되는가를 파악한다(12.6).
- 창문관리기의 역할을 살펴 본다(12.7).
- 공통탁상환경(Common Desktop Environment:CDE)에서 표준들에 대하여 배운다(12.8).
- xterm의뢰기를 리용한다(12.9).
- X프로그람들과 함께 리용되는 일반지령행선택항목들을 배운다(12.10).
- 마우스와 xclipboard를 리용하여 서로 다른 창문들사이에 자료를 복사한다(12.11).
- xclock, xcalc, xkill, xload와 같은 일반적인 의뢰기를 리용한다(12.12).
- xinit에 리용되는 시동파일과 .xinitrc파일을 구성한다(12.13).
- X자원을 지정하고 그것들을 xrdb와 -xrm 선택항목으로 재정의한다(12.14).

주해 이 장은 일부 창문체계 즉 애플, Windows, 직관물관리기(Presentation Manager)에 대한 지식을 전제로 한다. 또한 이전의 장에서 얻은 TCP/IP 에 대한 지식을 리용해야 한다. 마우스로 객체, 강조된 본문을 선택하거나 창문을 움직이고 크기를 재설정하는 방법은 알고 있다고 생각한다.

12.1 X를 왜 리용하는가

우리는 왜 도형사용자대면부를 리용해야 하는가? 류행이기때문에 X를 리용한다는것은 무의미하다. 왜 그렇게 해야 하는가 하는 기본원인을 밝혀야 한다.

파일을 제거하고 싶다고 하자. 그렇게 하려면 rm foo를 리용할수 있다. 그러나 자기앞에 그 파일의 아이콘(작은 그림)이 보인다면 그 파일의 아이콘을 찰칵하고 마우스로 집어서 휴지통에 끌어다 놓을수 있다. 이 방법이 좀더 빠른 방법이다.

그러나 통용기호에 의한 표현인 *.txt에 해당한 파일묶음이라면 어떻게 하겠는가? 이 경우에 UNIX 지령행은 명백하다. rm *.txt는 파일들을 선택하고 휴지통에 끌기하는것보다 더빨리 입력될수 있다.

X를 리용하는가 지령행을 리용하는가는 시간이나 노력적으로 적당히 비교해서 결심해야 하는것이다. 어쨌든 X에서 하는것이 더 쉬우면 그 환경을 리용할것이다. 지령행으로 파일과 등록부를 관리하는것은 때로 곤난하다.

그러나 도형을 보려고 할 때는 지령행이 해답을 주지 못한다. 전통적으로 UNIX말단들은 점들의 모임이 한 문자를 이루는 문자현시장치들이였다. 말단들은 건반우의 모든 건들을 표현하는 고정된 문자모임과 몇개의 칸그리기문자들을 처리할수 있다. 말단은 대표적으로 수평으로 80개의 문자, 수직으로 25개의 문자를 현시한다.

그러나 그림은 매점을 개별적으로 조작할것을 요구한다. X는 비트배렬현시장치(bit-mapped display)를 리용하는데 여기서는 화면우에 이러한 점들이 여러개 있을수 있다. 이것은 일반적으로 해상도가 800×600점인 화면인데 수평으로 점이 800개 있다는것을 의미한다. 매점을 조종하기 위한 능력과 함께 다중점들의 리용가능성은 X에 도형을 처리할수 있는 힘을 준다. 그러므로 X를 실행시키려면 자기의 기계와 VGA기판에 될수록 많은 기억기가 필요하다.

X를 리용하는 세번째 원인이 있다. 응용프로그람들이 이 가동환경으로 급속히 이행되고 있으며 X의 응용프로그람들이 홍수처럼 쏟아 져 나오고 있다. X를 모르면 Netscape 즉 인터네트와 Web에 접근하는데 리용되는 도구를 쓸수 없다.

12.2 X에서의 도형사용자대면부

Microsoft Windows와 같은 어떤 창문체계(GUI)를 리용한다면 다중창문으로 작업할것이다. 그것들은 개별적인 응용프로그람들을 표현한다. 다중창문을 가진다는것은 사용자가 앞에 다중말단들을 가진다는것과 같은데 거기서는 한 응용프로그람으로부터 다른 응용프로그람에로 자료를 옮길수 있다. 작업하는 창문의 수는 화면의 크기와 그것을 조종할수 있는 능력에 따라 제한된다.

X도 창문체계이며 자기의 GUI를 가지고 있다. 그러나 마이크로쏘프트나 애플의 창문체계와는 달리 X의 GUI에서 보고느끼기(look and feel)는 표준적이지 못하다. X의 설계가 형태별로 응용프로그람을 분리하기때문에 여러가지 형태로 이것들을 볼수 있다.

X를 기동시킨후에 어떤 개별적인 프로그람을 실행시켜 즉시에 창문형태를 변화시킬수 있다. 이 프로그람을 **창문관리기**라고 부르며 X에는 그러한 창문관리기가 여러개 있다.

12.3 X의 시동과 정지

X를 시동시키는데는 기본적으로 두가지가 있다. 한가지 방법은 xdm지령으로 자동적으로 시동시키는 것이다. 이 지령은 init(PID 1을 가진 프로쎄스)에 의하여 /etc/inittab로부터 기동시간동안에 실행된다. 그때 체계를 입력하기 위하여 사용자이름과 통과암호를 입력하여 자기를 인증해야 하는 도형방식의 화면이 현시된다. 그렇지 않으면 xdm은 뿌리사용자등록자리로만 실행될수 있다.

체계를 혼자서 사용한다면 startx지령으로 X를 호출한다.

```
startx
```

이 지령은 X봉사기를 기동시키며 탁상화면에 아이콘들을 넣는다. 이 아이콘들을 찰칵하지 않으려면 말단을 모의한 xterm의뢰기(X Window체계에서 흔히 리용되는 의뢰기)를 호출함으로써 작업을 시작할 수 있다.

X를 초기화할 때 startx를 리용하지 않고 xinit지령을 리용할수도 있다.

```
xinit
```

이 지령도 X봉사기를 기동시킨다. X를 아직 전용화하지 않았다면 이 지령은 xterm창문을 화면의 왼쪽웃구석에 설정한다. xinit의 동작은 구성파일 .xinitrc에 의해 결정되는데 후에 론의한다.

X에서 탈퇴하는 표준적인 방법은 없다. 다음의 두가지 방법들이 대체로 리용된다.

- 탁상화면의 아무곳이나 화면의 끝자리들에 유표를 놓고 마우스단추를 누른다. 차림표가 펼쳐 지면 Exit Session 혹은 Log out와 같은 항목이 있는가를 찾아 본다.
- xinit로 시작한 경우에는 X를 기동시킬 때 나타나는 xterm창문에 있는 exit지령을 리용해야 한다.

다음절에서는 x를 특징 짓는 의뢰기-봉사기구조의 차이와 TCP/IP망에서 X를 적합하게 실행하도록 어떻게 구성되였는가를 리해하여야 한다. 이에 대해서는 다음부분에서 론의한다.

12.4 X의 구성방식

X는 단순한 창문체계가 아니라 완전히 혁신적인 제품이다. MIT가 자기의 계획에 관하여 작업을 시작하였을 때 그들은 사람들이 여러가지 형태의 도형현시장치로 작업하고 있다는것을 깨달았다. 매 도형현시장치는 선과 도형을 화면에 그리는 자기의 방식을 가지고 있다. 어떤 응용프로그람을 이 모든 현시장치에 쓰려면 매 현시장치에 대한 응용프로그람의 여러가지 판본이 있어야 한다.

100가지 형태의 현시장치에 대해서 100개의 프로그람판본을 설계한다는것은 어리석은 일이다. 그리하여 MIT에서는 현시장치가 어떤 특별한 프로그람에 의해서 조종되여야 한다고 생각하였다. 응용프로그람은 현시장치조종프로그람에 자기의 출력을 보낼것이다. 이것은 응용프로그람을 의뢰기와 봉사기의 구성요소로 분할하도록 이끈 중요한 계기였다.

Windows응용프로그람에서는 PC의 현시장치를 조종하는 일은 그 의뢰기응용프로그람자체가 수행한다. X는 이 방식과 반대이다. X의 구성방식은 **봉사기**에 현시장치를 조종할 책임을 주는것이다. 그 응용프로그람자체는 **의뢰기**로서 실행된다. X봉사기와 의뢰기들은 같은 기계에 있을수 있지만 X는 그것들이 분리되여 있을 때 빛이 나며 초기에는 TCP/IP망에서 실행되도록 설계되였다. 그 전송은 전적으로 믿음직하며 TCP/IP통신규약(23.1)을 리용한다.

그러면 X봉사기는 무엇을 가지고 있는가? 이 대답을 유도하려면 고급한 계산능력때문에 또는 간단히 자기 기계에 그러한 능력이 없기때문에 원격기계상에서 그 프로그람을 실행시키려 한다고 생각해야 한다. X기술은 사용자가 원격기계에서 X의뢰기를 실행시키게 하며 자기의 국부기계에 현시를 하게 한다.

봉사기가 현시장치를 조종한다면 봉사기는 또 현시장치의 모든 입력을 감시하여야 한다(실례로 수산기프로그람의 입력). 이것은 봉사기가 마우스와 건반까지도 포함한다는것을 의미한다.

이것은 X에서 중요한 의미를 가진다. X프로그람은 창문그리기의 복잡한 과제로부터 벗어 나 아주 간편해 졌다. 서로 다른 말단에서 X의뢰기를 실행시키기 위해서 사용자가 해야 하는 일이란 그 말단을 위한 X의 봉사기요소에 쓰기를 하는것뿐이다. 일단 쓰기되면 X프로그람은 앞으로 이 새로운 봉사기프로그람에 쓰기할수 있다.

주해

X현시장치는 마우스와 건반을 포함한다. 또한 여러개의 감시기상에 다중화면을 구성할수 있다. 사실 대부분의 UNIX체계는 다중화면을 제공한다.

12.5 X프로그람을 원격적으로 실행시키기

단일사용자체계에서는 봉사기와 의뢰기가 같은 기계에서 실행되는데 사용자는 현시장치를 조종하는데 대해서 조심하지 않아도 된다. 이러한 상황은 망에서 X를 실행시킬 때는 완전히 달라 진다. 어떤 사람이 자기의 기계(saturn)에서는 불가능하지만 원격기계(uranus)에 있는 수산기프로그람 xcalc를 리용하고 싶다고 하자. 그 사용자는 프로그람실행은 거기서 하지만 자기 건반으로 자료를 입력해서 현시는 자기 화면에 현시하고 싶을것이다. 이렇게 하기전에 해야 할 두가지 조건이 있다.

- 자기의 기계 saturn에 있는 봉사기는 다른 기계가 자기의 현시장치에 쓰기 할수 있게 되여야 한다.
- 원격기계 uranus의 의뢰기프로그람은 saturn의 현시장치에 자기의 출력을 쓸수 있도록 방향이 정해 져야 한다.

첫번째 요구는 xhost지령을 리용할 때 발생되고 두번째 요구는 DISPLAY변수를 설정한다든가 의뢰기와 함께 -display선택항목을 리용할 때 제기된다. 우리는 앞으로 나가면서 이 두가지 특징에 대하여 볼것이다. 후에 xterm에 대하여 론의하지만 모든 X의뢰기들은 xterm이 제공하는 UNIX지령행으로부터 실행된다는것을 알아야 한다.

참고

X를 기동시킨 상태에서 탁상화면에서 xterm창문을 찾지 못한다면 $HOME/.xinitrc파일안에 xterm & 항목을 넣고 X를 재시동시키시오. 이 파일은 12.13에서 론의된다.

12.5.1 현시장치에 대한 쓰기허용(xhost)

원격프로그람을 국부적으로 실행시키려면 우선 X봉사기가 자기의 기계에서 실행되게 하여야 한다. startx와 xinit지령들을 쓰면 X가 기동되게 할수 있다. 이제 자기의 국부기계에 있는 xterm창문으로부터 xhost지령을 리용하려고 한다.

```
$ xhost +uranus
uranus being added to access control list
```

uranus상에 있는 사용자는 자기의 현시장치에 쓰기할수 있다. 사용자는 -기호로 xhost설정을 해제할수 있으며 +기호로 모든 기계들이 자기의 현시장치에 쓰기가능하게 할수 있다.

```
$ xhost -               다른 사용자가 현시할수 없다
access control enabled, only authorized clients can connect
$ xhost +               다른 사용자가 현시할수 있다
access control disabled, clients can connect from any host
```

xhost +uranus 혹은 xhost +로 자기의 현시장치가 쓰기가능한가를 확인한다. 이 책에서 원격적으로 지령을 실행시키는 두가지 기술을 서술한다. 첫번째는 telnet로 하는것이고 두번째로는 rsh로 하는것이다.

참고

자기의 현시장치에 쓰기허락된 기계들의 목록을 가지고 있다면 매번 DISPLAY를 설정하는것보다 차라리 /etc/xn.hosts파일(n은 현시장치의 개수이다.)에 주콤퓨터이름들을 보관하는것이 더 좋다.

12.5.2 출력을 어디로 보낼것인가(DISPLAY변수)

우리는 telnet가 원격프로그람을 실행시켜 준다는것을 잘 알고 있다. 아래에서는 uranus에 《telnet》지령을 준다.

```
$ telnet uranus
Trying 192.168.0.4...
Connected to uranus.
Escape character is '^]'.
login: henry
Password: *******
```

프롬프트가 현시되면 새로운 쉘안에 있는 DISPLAY변수를 정의하고 보낸다.

```
DISPLAY=saturn:0.0              saturn은 /ectc/hosts에 있어야 한다
export DISPLAY              C쉘문법에 대해서는 17.2를 보시오
```

DISPLAY변수의 출력은 <주콤퓨터이름:봉사기.화면>이라는 형식으로 되여 있다. saturn은 주콤퓨터이름이며 :0은 X봉사기프로그람의 실체이며 .0은 현시장치의 화면번호이다. 봉사기와 화면은 보통 단일사용자체계에 대해서 0이다. 그리고 화면번호(.0)은 생략된다. saturn이 /etc/hosts파일에 정의되여 있지 않으면 그곳에 있는 IP주소를 리용해야 한다.

우의 설정은 어떤 uranus에 있는 X의뢰기가 대체로 자기것보다 saturn기계의 현시장치를 리용한다는것을 의미한다. 이제 uranus상에서 xcalc프로그람을 실행시킬수 있으며 자기의 기계에 출력을 현시할수 있다.

```
xcalc &
```

수산기는 그 사용자의 기계에 현시되였으며 따라서 모든 계산을 할수 있다. X는 건반이 봉사기정의에서 구성요소로 되고 이 봉사기는 국부적으로 실행되기때문에 원격의뢰기와 함께 국부건반을 리용하게 한다.

12.5.3 rsh를 -display선택항목과 함께 사용하기

telnet(사용자가 등록가입한 다음 거기서 지령을 사용하게 하는)를 사용하는것대신에 rsh를 리용하여 원격기계우에서 지령을 실행시키고 출력을 국부기계에 현시할수도 있다.

거기서는 모든 X의뢰기가 제공하는 -display선택항목을 리용해야 한다. 이 선택항목은 현시장치이름(주콤퓨터이름:복사기.화면형식)뒤에 놓인다. 아래에 자기의 국부적인 현시장치를 리용하면서 uranus에서 꼭 같은 xcalc의뢰기를 실행시키는 방법을 보여 준다.

```
rsh uranus xcalc-display saturn:0              화면번호는 빠졌다
```

이것은 지령을 원격적으로 실행시키는 아주 효과적인 방법이다. 필요하다면 xcalc의 절대경로를 리용한다. 그러나 다음과 같이 xterm의뢰기를 실행시킬 때는 아주 중요한 결론이 제기된다.

```
rsh uranus Xterm -display saturn:0
```

xterm창문은 saturn의 현시장치에 나타나지만 xterm창문으로부터 실행되는 어떤 프로그람은 -display선택항목을 리용하지 않고 같은 현시장치에 계속 쓰기한다. 이 결론은 분명하지는 않지만 사실이다. 그러나 이것을 가능하게 하려면 rsh(11.10)를 사용할수 있는 해당한 권한을 가지고 있어야 한다.

주해

telnet나 rsh로 실행되는 원격프로그람은 그 출력이 국부기계에 나타난다 해도 원격기계안에 파일들을 보관한다. rsh가 표준출력장치를 리용하는 문자방식의 의뢰기와 함께 리용될 때 원격적으로 출력을 보관하려면 >문자를 \에 의하여 해제했는가를 확인하여야 한다. 이에 대해서는 이미 11.9에서 론의되였다

12.6 X기술과 구성요소

X에서 전체 화면(탁상화면)은 보통 다중창문으로 가득차게 된다(그림 12-1). 그 어떤 GUI에서나 항행도구는 마우스이다. 모든 마우스들은 2개 혹은 3개의 단추를 가지고 있으며 대체로 왼쪽 단추가 자주 리용된다. 마우스지시자들은 어떤것을 찰칵하거나 그것을 끌기하는데 리용된다. 편집기에서 어떤 본문을 지시자로 찰칵하면 지시자는 I모양의 유표로 변한다. 그다음 본문우에서 마우스왼쪽단추를 눌러 끌기하면 그 본문은 강조된다.

차림표는 탁상화면(**뿌리창문**이라고도 한다.)에 때로는 화면의 가장자리들중의 한 곳에 마우스단추로 찰칵하면 펼쳐 진다. **뿌리차림표**라고 부르는것이 있는데 그 차림표는 력서와 우편,Web열람기,말단과 같은 중요한 X응용프로그람이나 X대화를 완료하기 위한 선택항목들을 제공한다.

탁상화면우의 모든 창문은 아이콘화(최소화된것)되지 않았다면 제목띠를 보여 준다. 창문을 아이콘화한다는것은 창문을 탁상화면에 자리 잡는 아이콘으로 변화시킨다는것을 말한다. 그림 12-1에서 탁상화면의 왼쪽 웃구석에서 아이콘화된 창문을 볼수 있다. 아이콘화된 창문을 찰칵하면 다시 원래대로 회복할수 있다.

가상적인 창문으로 작업하기전에 **입력초점**이 그우에 있는가를 확인해야 한다. 다시 말하여 창문은 건반입력이 그 응용프로그람에 도달하도록 선택되여야 한다. 선택된 창문은 제목띠색이 차이나는데 언제나 한개 창문만이 이 색을 가진다.

Windows에서는 창문의 아무곳에 마우스로 찰칵하여 선택한다. X에서 창문선택기술은 리용되는 창문관리기에 관계된다. CDE와 Motif는 Windows체계에 따른다. 그러나 어떤것들은 사용자가 제목띠에 찰칵할것을 요구한다.일부 체계들에서는 창문을 선택하려면 찰칵하지 않고 유표를 창문우에 움직이면 된다.

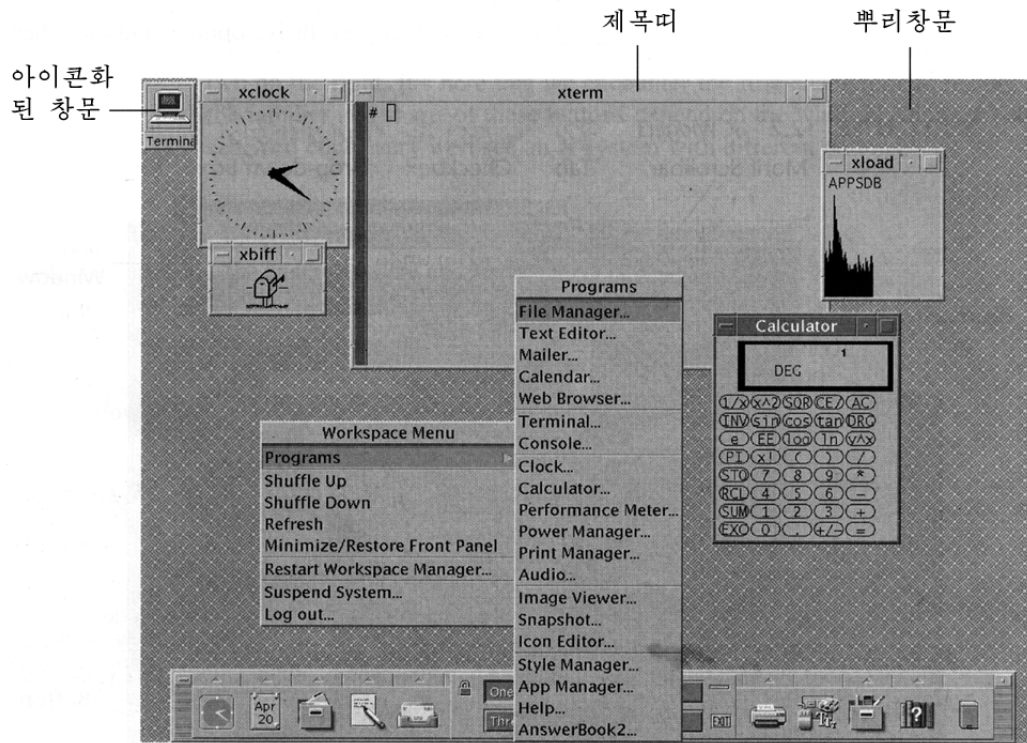



그림 12-1. X의 탁상화면

창문이 보이지 않는다면(창문탄창의 밑준위에 있다.) Windows형태인 [Alt] [Tab]를 리용하여 창문을 우로 끌어 올릴수 있다. 사용자는 이런 방법으로 창문의 일부분이라도 언제나 보이게 위치를 정할것이다. 창문을 움직이려면 제목띠를 찰칵하고 그것을 끌기해야 한다. 또한 네 모서리의 아무곳을 끌기하여 창문크기를 조절할수 있다. 이 모든것을 할수 있다면 X에서 작업할수 있는 준비가 되여 있다.

참고

Windows사용자들은 X가 창문을 움직이는 추가적인 방법을 가지고 있다는것을 알것이다. 그 방법은 창문을 선택하지 않고도 움직일수 있는것이다. 즉 [Alt]건을 누르고 창문의 제목띠를 찰칵한다. 그리고 창문을 움직인다. 그러면 창문을 선택하지 않고도 움직일수 있다.

X의 부분품들

X는 **부분품**(widget)이라고 부르는 구성요소들로 동작한다. X의 부분품들은 단추, 흘림띠, 검사칸 그리고 마우스찰칵에 응답하는 객체들로 구성되여 있다.

그림 12-2에서는 Netscape의 구성창문이 2개 겹놓아 져 있는데 그안에 있는 구성요소들을 보여 준다. 왼쪽에서 흘림띠를 볼수 있다. 현재 보이지 않는 본문을 현시하기 위하여 마우스로 끌기한다든가 창문을 찰칵할수 있다. 서로 상반되는 항목들을 보여 주는 3개의 단일선택단추묶음들을 볼수 있다.

사용자는 검사칸에서 하나 혹은 그이상의 항목들을 선택할수 있다. 항목을 선택하려면 찰칵표식을 추가하거나 작은 칸이나 다이아몬드표식이 눌리워 지지 않게 한다. 표쪽(tab)과 내리펼침칸(drop-down box)도 있다.

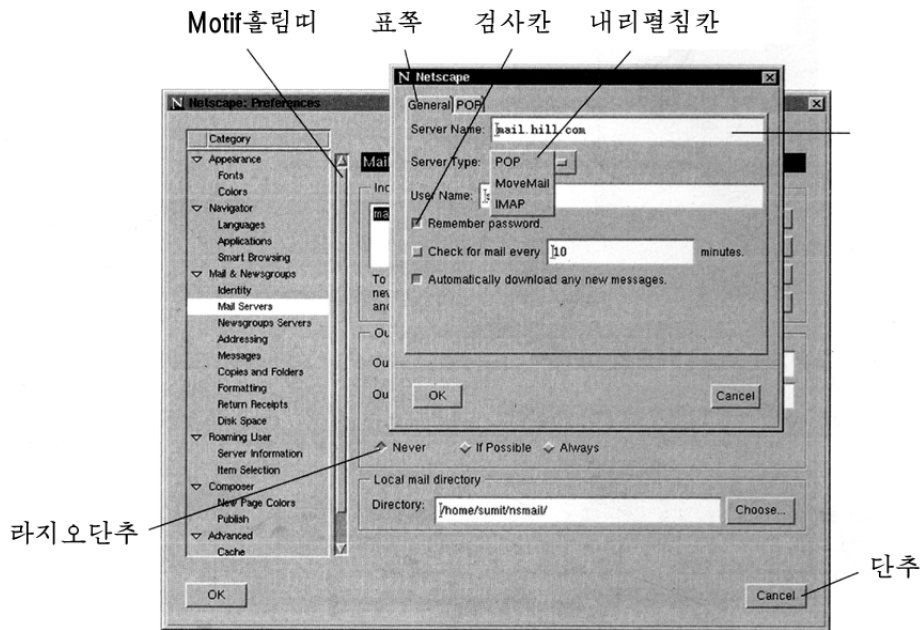



그림 12-2. X의 부분품들

12.7 특수한 의뢰기로서의 창문관리기

X의뢰기들은 화면의 개별적인 창문에 자기의 출력들을 현시하지만 X봉사기는 그것들을 위한 그 어떤 틀이나 차림표를 제공하지 않는다. 그렇다고 의뢰기들도 그러한것들을 제공하지 않는다. 더우기 《틀이 없는》 창문을 움직이고 그 크기를 조절할수 있는 방법도 없다. 그러면 이러한 기능을 누가 제공하는가? X의뢰기들의 보고느끼기는 특별한 X의뢰기인 **창문관리기**에 의해 결정된다.

창문관리기는 모든 X응용프로그람들에 수많은 조작을 제공한다. 창문관리기는 의뢰기프로그람우에 내장되여 있으며 모든 의뢰기들이 제작자들에 관계없이 같은 형태를 가지게 한다. 우스운것은 모든 의뢰기에 대한 창문관리기능이 또 다른 의뢰기에 의해 수행된다는것이다. 봉사기는 이러한 역할을 전혀 하지 못한다. 창문관리기는 어떤 틀과 단추들을 제공하여 창문들을 움직이고 크기를 조절할수 있게 한다. 또한 창문차림표도 제공한다.

이전에는 twn, Open Look(olwm)과 Afterstep와 같은 창문관리기들도 있었다. 최근에는 표준적인 창문관리기로서 mwm(Motif window manager)과 dtwm(일반탁상화면환경에 의하여 리용되는 창문관리기)도 있다. dtwm은 IBM의 직관물관리기와 류사하게 설계된 Motif에 기초한다. Linux도 자기의 표준관리기들인 fvwm,fvwm95(Windows에 기초한), kwm(CDE에 기초한)들을 가지고 있다.

그림 12-3에 xterm의뢰기(사용자에게 쉘프롬프트를 주는 의뢰기)를 포함하여 dtwm창문관리기를 보여 준다. 제목띠는 왼쪽에 한개와 오른쪽에 2개의 단추를 가지고 있다. 여기에 X의 특징적인 기능이 있다. 이 단추들의 개수와 모양은 사용자가 선택한 창문관리기에 의존된다. 사용자는 서로 다른 단추들을 가진 X체계를 볼수 있을것이다.

제목띠의 오른쪽에 있는 두개의 단추들중의 하나는 정방형으로 표현되는데 창문을 최소화하는데 리용된다. 즉 창문을 아이콘으로 변화시킨다. 옆의 다른 단추는 초기에는 볼록 나온 4각형으로 나타나는데 창문을 최대로 하는데 리용된다. 일단 창문이 최대로 되면 4각형은 오목 들어 간 상태로 된다.

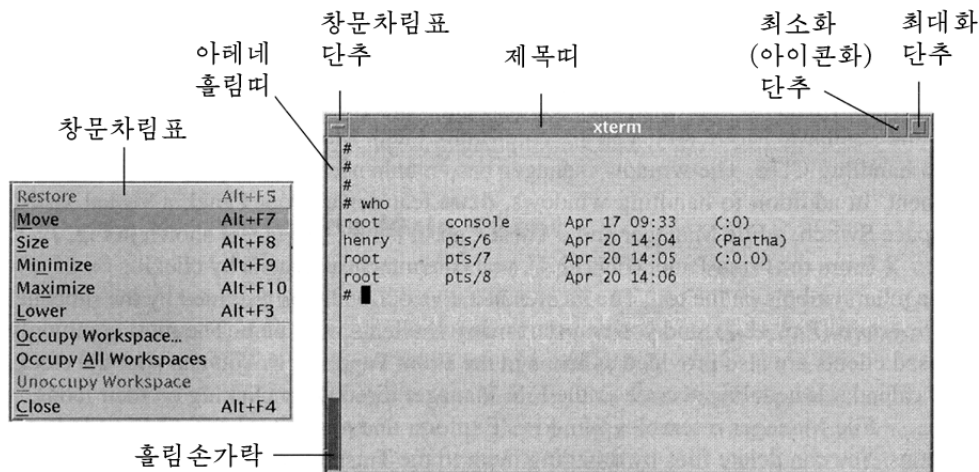



그림 12-3. X term창문과 dtwm창문차림표

왼쪽단추는 완전히 다른 기능을 가지고 있다. 사용자가 그 단추를 찰칵하면 창문차림표가 펼쳐 진다. 우에서 취급한 대부분의 창문관리기능들이 이 창문차림표에 의해 조종될수 있다. 오른쪽단추들은 보이지 않지만 왼쪽의 단추가 보인다면 창문의 최소화, 최대화, 닫기, 움직이기 그리고 크기조절까지 하는 차림표항목을 리용할수 있다. X를 어떻게 기동시키든지간에 사용자는 창문관리기가 X에 의하여 리용되는 어떤 구성파일로부터 실행된다는것을 확인하여야 한다. 그렇게 하지 않으면 X를 기동시킨후 xterm창문으로부터 직접 창문관리기를 실행할수 있다. Motif창문관리기를 리용하고 있다면 배경에서 의뢰기를 실행시킨다

```
mwm &                    적당한 창문관리기프로그람을 리용한다
```

X를 기동시킨 상태에서 빠진 창문틀을 찾는 경우에만 xterm창문에 이 지령을 입력한다. 현재 존재하는 창문들과 앞으로의 모든 창문들은 그림 12-3에서 xterm창문안에 보여 준것들과 류사한 보기특징을 가질것이다.

12.8 공통탁상환경

앞으로는 임의의 X환경에서도 사용자가 편리하도록 개발이 진행될것이다. 이전에는 Motif가 표준창문관리기로서 개발자들에게 가장 널리 적용되였다 해도 X의 제품들은 여전히 개발자에 따라 달랐다. 탁상화면의 형식도 단일하지 않았으며 응용프로그람을 리용하는 방법도 또한 서로 달랐다. 이것은 Motif를 리용한 제품들이라 해도 서로 다른 X판본들에 대한 사용자들의 요구를 복잡하게 만들었다. Motif는 단순한 《탁상환경》이 전체 화면의 보고느끼기를 변화시키기전에는 오래동안 표준으로 남아 있었다.

주요회사들인 IBM, HP, 싼쏘프트, 노벨은 Motif에 기초하여 1993년에 COSE(Common Open Software Environonment)운동을 시작하였다. 그들은 X를 실현하는 모든 제작자들의 제품들이 탁상화면은 물론 모든 창문들도 꼭 같은 보기와 느끼기를 가지도록 모든것들을 담보하는 표준적인 탁상화면 즉 **공통탁상환경**(Common Desktop Environment:**CDE**)을 도입하였다.

CDE에 리용되는 창문관리기는 dtwm(desktop window manager)이다. CDE는 Motif와 아주 류사하다. 그래서 Motif에 완전히 익숙되였다면 CDE조종이 좀 복잡하다는 감을 느끼게 된다. 창문관리기는

이 환경에서 해야 할 일이 많다.

dtwm은 창문조종외에도 정면판(Front Panel), 가상작업공간스위치, 파일관리기와 휴지통(Trash Can)의 기능도 가지고 있다. 그림 12-1에 CDE를 보여 준다.

정면판에서(그림 12-4) 3각형기호를 찰칵하여 보조판들을 꺼내 볼수 있다. 가장 일반적인 선택항목들은 보조판프로그람들이 제공하는데(그림 12-1) 여기로부터 대부분의 X의뢰기가 실행된다. 가장 일반적으로 리용되는 의뢰기들은 정면판에 아이콘으로도 제공된다. 즉 시계, 력서(calendar), 파일관리기들의 아이콘을 찰칵함으로써 시작할수 있다.

파일관리기는 Windows Explorer와 류사하며 기본파일조작기능을 수행한다. 파일들은 휴지통에 끌기함으로써 지울수 있다. 이런 방법으로 지워 진 파일들은 후에 재생될수 있다.

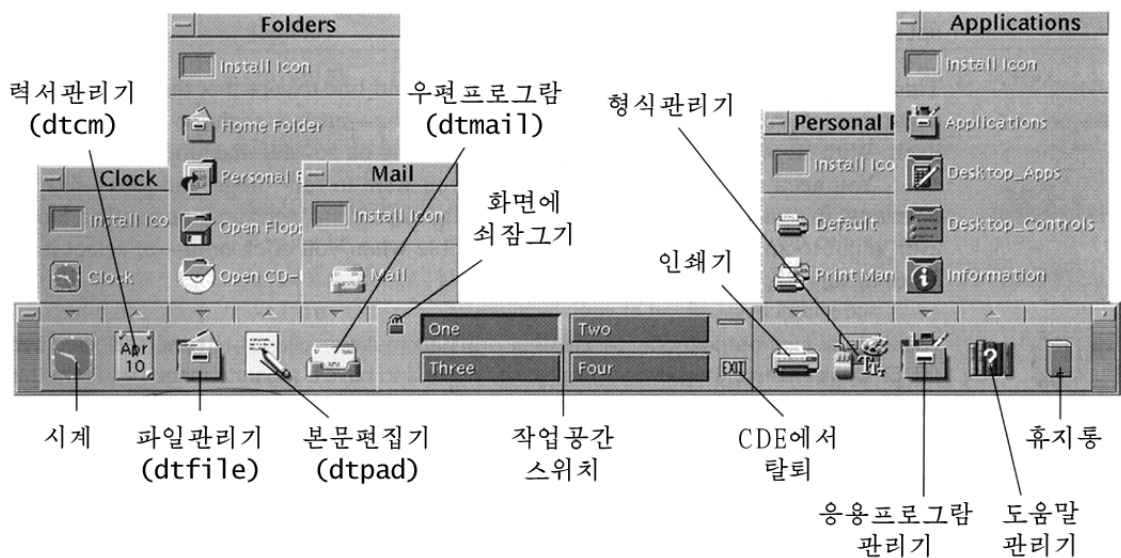



그림 12-4. CDE정면판

정면판의 중심에 작업공간스위치(Workspace Switch)가 있다. 이것은 자기의 기계우에 다중탁상화면을 만들게 하는 4개의 단추들로 이루어 져 있다. 어떤 탁상화면상에서 한개의 Windows를 선택할수 있으며 또 다른것도 선택할수 있다. 그리고 관계되는 작업공간단추를 찰칵함으로써 자기가 요구하는 탁상화면을 선택할수 있다. 즉 4개의 단추로 4개의 탁상화면을 가질수 있다. 따라서 크게 4개의 화면을 가지고 있는것처럼 보인다.

앞으로 CDE에 대한 론의는 더 진행된다. 다음에는 X의 기능에 대해서 론의한다. 때때로 창문관리기로서 Motif를 가정한다. 독자들은 서로 다른 창문관리기나 다른 탁상화면환경을 리용하는데 따라 자기의 체계상에서 Windows와 그 탁상화면의 형식이 다를수 있다는것을 기억하시오.

Linux도 자기의 탁상화면을 가지고 있었는데 지금은 두가지 제품 KDE와 GNOME(GNU로부터 나옴)를 가지고 있다. KDE는 창문관리기로서 kwm을 가지고 있다. CDE와 같이 KDE도 판(CDE의 정면판), 가상탁상화면, K파일관리기, 휴지통의 기능을 가지고 있다. 그러나 GNOME가 더 인기를 모으고 있으며 표준으로 될 가능성도 높다.

12.9 주의리기(xterm)

앞에서 언급한것처럼 xterm은 가장 중요한 X의뢰기이다. 기정적으로 xterm은 X가 xinit로 기동될때 나타난다(그림 12-3). 보이지 않은 창문들을 찾으려면 배경에서 창문관리기를 불러 내는것보다 이 xterm창문에서 조작하는것이 더 좋다(12.7).

xterm은 문자방식의 말단처럼 동작하는데 단순히 쉘의 실체이다. 모든 UNIX지령들과 다른 X프로그람들을 실행시킬수 있는 이 창문에서 쉘프롬프트가 현시된다. 배경에서 여러개의 일감들을 실행시키려면 &를 뒤에 붙여 준다.

```
xclock &
xfm &                파일관리기
netscape &
```

이것은 매개 응용프로그람이 겹놓인 창문으로 보이는것을 제외하고는 지령행우에서 리용하는것과 같은 기술이다. 사용자는 여러대의 기계상에서 X와 작업하는 느낌이 들것이다.

vi나 emacs와 같이 전경일감들을 실행시키고 싶다면 다중xterm창문이 필요하다. 그러면 그 창문에서 프로그람을 편집할수 있고 다른 곳에서 그것을 실행시킬수 있다. 그러나 xterm창문의 입력은 그 창문을 찰칵하거나 [Alt][Tab]로 선택할 때만 가능하다.

주해

탁상화면환경은 고급한 말단모의의뢰기 즉 cdterm(CDE)과 kvt(KDE)를 제공한다. xterm과 달리 그것들은 언제나 흘림띠를 현시하며 개별적인 차림표띠를 가지고 있다. 이 차림표로부터 폰트크기와 색을 설정할수 있으며 본문을 복사하고 붙일수도 있다.

주의

X의뢰기지령에 &를 붙이지 않으면 그 지령은 xterm창문의 새끼창문을 만든것으로 된다. 이 어미창문을 없애면 새끼창문도 완료된다. 이것은 보통 사용자가 원하는것은 아니다. 따라서 모든 X의뢰기들은 배경에서 즉 비동시적으로 실행되여야 한다.

흘림띠의 사용

xterm지령으로 흘림띠의 기능을 능동으로 할수 있다. 이전의 지령이나 그 출력을 다시 호출하려면 -sb 그리고 -sl선택항목과 함께 xterm지령을 리용한다.

```
xterm -sb -sl 10000 &                마지막 10000행을 보관한다
```

보관된 행의 수는 -sl선택항목에 의해 결정된다. 이때 아테네흘림띠(Athena scrollbar:MIT가 개발한 초기의 흘림띠)가 xterm창문의 왼쪽에 나타난다. 이 흘림띠에는 회색으로 된 흘림손가락(thumb:흘림칸)이 있는데 흘림띠는 지령들이 입력되면 크기가 줄어 든다. 과거의 지령들을 다시 보려면 가운데단추로 흘림손가락을 끌기해야 한다.

12.10 지령행선택항목

대부분의 X의뢰기프로그람들은 매우 많은 선택항목들과 함께 실행된다. 대부분의 선택항목(표 12-1)들은 실지로 모든 지령들에 공통적이다. 그것들은 자주 략자로 쓰이는데 략자는 표에서 괄호안에 보여준다. 우리는 그 의미에 대해서 간단히 론의할것이다.

표 12-1. X의뢰기들이 리용하는 공통지령행선택항목들

선택항목	의 미
-display	주콤퓨터(그우에서 의뢰기가 출력을 현시한다.)의 현시를 지정한다
-gemetry	창문의 크기와 위치(-g)
-fore ground	창문의 전경색(-fg)
-background	창문의 배경색(-bg)
-reverse	전경색과 배경색을 서로 바꾼다(-rv)
-iconic	최소화된 형식으로 의뢰기를 시동시킨다
-title	제목띠에 현시될 창문이름
-name	응용프로그람의 이름
-xrm	자원파일들의 설정들을 재정의하는 자원명세

12.10.1 창문위치와 크기(-geometry)

어떤 선택항목도 없이 X의뢰기를 실행시키면 그 창문은 기정적인 위치와 크기로 설정된다. 탁상화면에 여러개의 의뢰기들을 배치할 필요가 있을 때 그것들을 쉽게 선택하려면 적당한 크기와 위치를 설정해주어야 한다.

대부분의 X의뢰기들은 의뢰기의 위치와 크기를 설정하기 위하여 -geometry선택항목을 제공한다. 이 선택항목뒤에는 두가지 요소 즉 창문의 크기와 화면의 매 경계로부터 그 창문모서리의 위치가 놓인다.

xterm창문에 대해서 창문의 크기는 문자방식으로 지적되고 위치는 점방식으로 지적된다. 다음의 지령행은 xterm을 창문크기는 40×12문자방식으로, 왼쪽웃구석모서리는 왼쪽면으로부터 10점 떨어 지고 우에서는 20점 떨어 진 위치로 설정한다.

```
xterm -g 40x12+10+20 &
```

다른 의뢰기들에 대해서는 40×12가 40픽셀×12픽셀 즉 매우 작은 창문을 의미할것이다. 기정적으로 xterm창문은 80×24문자방식의 크기를 가진다. 그리고 화면의 서로 다른 령역에 다중xterm창문들을 가질수 있다. 크기는 생략될수도 있다.

```
xterm -g -10-20 &        화면의 오른쪽밑모서리로부터의 변위
xterm -g +10-20 &        화면의 왼쪽밑모서리로부터의 변위
xterm -g -10+20 &        화면의 오른쪽웃모서리로부터의 변위
```

두 수가 다 령이면 창문은 화면의 모서리에 배치된다(-g -0 -0은 오른쪽밑모서리에 창문을 배치한다).

12.10.2 색설정(-fg, -bg, -rv)

-foreground(-fg)와 -background(-bg)선택항목은 의뢰기의 색을 결정한다. 다음의 형식으로 xterm창문에 색을 설정할수 있다.

```
xterm -fg darkslategrey -bg lightblue &
```

X는 /urs/lib/xivrgb.txt 본문파일안에 색자료들을 가지고 있다(이 위치는 변할수도 있다). 이 파일은 매행이 적, 록, 청의 세가지 색의 량을 표현하는 세개의 수로 되여 있는 700가지 색(RGB값)서술을 가지고 있다. 여기에 그 파일안에 있는 대표적인 일부 항목들을 보여 준다.

```
255  255  240        ivory
```

```
 47   79   79        dark slate gray
 47   79   79        DarkSlateGray
  0    0  128        navy blue
135  206  235        sky blue
  0  255  255        cyan
224  255  255        light cyan
```

전경색과 배경색은 -reverse(-rv)선택항목으로 반전될수 있다. 이 선택항목은 창문(어미창문을 말한다.)을 나타내고 싶을 때 여러모로 편리하다. xterm창문을 구별되는 색으로 설정하면 자기도 모르게 창문을 제거하는 현상을 미리 막을것이다.

12.10.3 기타 선택항목

아이콘으로 시작하기(-icon)

-iconic선택항목을 써서 xterm의뢰기를 아이콘형식으로 실행시킬수 있다. 이 선택항목은 기본창문에 공간이 부족할 때 특별히 필요하다. 창문을 찰칵(창문관리기에 따라 한번 혹은 두번 찰칵한다.)함으로써 능동으로 만들수 있다. 실수로 xterm어미창문을 제거하지 않게 하기 위하여 아이콘화해서 보관할수 있다.

제목을 주기(-title)

제목띠에 현시될 제목선택항목(-title)을 제공한다. xterm의 여러개의 실체들이 실행될 때 그 실체들의 이름을 분리하는것이 더 좋다. 그 실체들이 실행되고 있는 기계라든가 그것들우에서 실행되는 응용프로그람의 뒤에 이름을 준다.

```
xterm -fg red -bg lightblue -title "sqlplus on uranus"
```

이름을 주기(-name)

-name선택항목은 봉사기가 식별할수 있는 이름을 제공한다. 독자들은 아이콘표식(label)안에 있는 이름을 보았을것이다. X자원들은 -name선택항목으로 제공된 문자렬을 리해한다. 이 원인으로 해서 -title선택항목보다 이 선택항목을 리용하는것이 더 낫다.

다른 두개의 선택항목 -bd와 -bw는 각각 경계색과 경계폭을 설정한다.

12.11 복사와 붙이기

xterm도 창문관리기(mwm이나 dtwm같은것)도 본문을 복사하고 움직이기 위한 어떤 특별한 차림표를 제공하지 않는다(cdterm은 Windows와 같이 복사, 자르기, 붙이기기능을 가지고 있는 차림표띠안에 Edit차림표를 현시한다). X는 오직 마우스단추를 리용해서만 본문을 복사하고 붙이는 조작을 제공한다. 그런 방법으로 본문을 어떤 창문에서 다른 창문으로 움직일수 있다. 실례로 vi와 emacs사이에 본문을 옮길수 있다.

본문을 복사하려면 왼쪽마우스단추로 그것을 끌기하여 강조시켜야 한다. 본문은 자동적으로 완충기안에 복사된다. 그다음 다른 창문을 왼쪽단추로 찰칵하고 본문을 붙이고 싶은 곳에 유표를 옮긴다. 그리고 가운데단추를 찰칵한다. 완충기안에 복사된 본문은 창문안에 삽입된다.

vi를 리용하는 경우에는 본문을 붙이기전에 자기가 입력방식에 있다는것을 확인해야 한다. 어떤 지령을 재실행하기 위하여 다른 xterm창문안에 있는 지령행을 복사하고 붙이기할수 있다.

끌기로 선택할 대신에 본문우에 여러번 찰칵하여 선택할수 있다. 두번찰칵은 단어를 선택한다. 세번 찰칵은 전체 행을 선택한다. X는 본문을 자르는 기능은 제공하지 않는다. 이것은 편집기의 기능을 리용해서 해야 한다.

참고

화면안에 있는 어떤 본문블로크를 선택하기 위하여 xterm을 리용한다면 왼쪽단추로 본문의 시작위치를 찰칵하고 본문의 끝에서 오른쪽단추를 찰칵한다. 그러면 전체 본문이 선택된다. 선택된 본문을 아무쪽으로 늘쿠려면 본문경계에서 오른쪽찰칵한다.

고급한 복사와 붙이기(xclipboard)

표준복사와 붙이기기능은 한계를 가진다. 한번에 오직 한개 단락의 본문밖에 선택할수 없다. 즉 후에 붙이기하겠다고 여러개의 단락을 꺼낼수는 없다. 이것은 마이크로쏘프트상에서 실행되는 쏘프트웨어에서도 제기되는 문제인데 사용자는 이 개개의 조작을 수행하기 위하여 두개 파일들사이를 반복적으로 왔다 갔다 하여야 한다. vi와 emacs는 이름 지어 진 완충기들로 이 문제를 조종하는데 X는 Macintosh와 같은 도구인 xclipboard프로그람으로 그 문제를 해결하였다.

xclipboard(그림 12-5)는 아테네부분품들을 리용하는 전형적인 MIT X응용프로그람이다. 이 창문의 꼭대기에는 6개의 단추가 있는데 마지막으로부터 2개는 초기에 회색으로 된다. 이 도구를 사용하려면 어떤 xterm창문으로부터 본문을 복사하고 그다음 그것을 xclipboard의 빈 창문에 붙이기한다. 오른쪽에 있는 계수기가 초기에 1을 보여 준다.

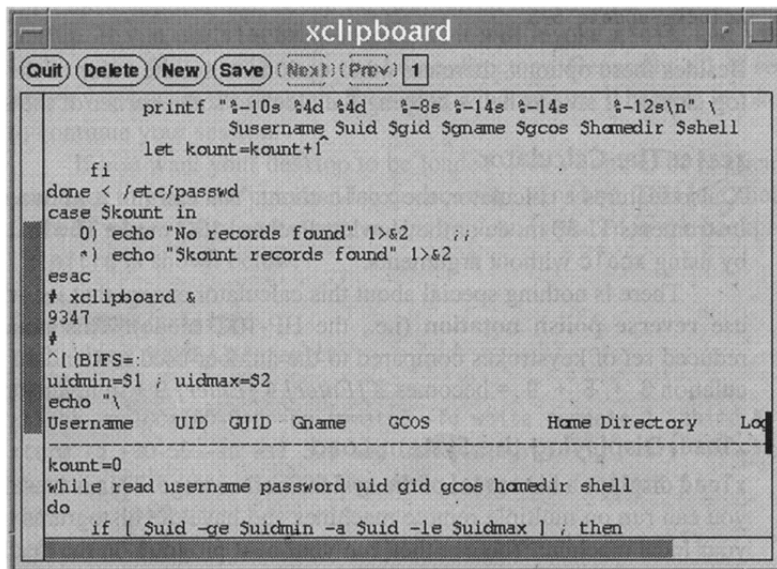



그림 12-5. xclipboard의뢰기

본문이 그 창문에 꽉 들어 찰만큼 너무 크면 아테네흘림띠가 나타나는데 가운데마우스단추로 흘림띠의 흘림손가락을 끌기할수 있다. xterm과 달리 xclipboard는 창문폭이 본문을 현시하기에 모자라면 수평흘림띠도 현시한다.

또 다른 본문토막복사조작을 반복하려면 먼저 New를 찰칵하여야 한다. 그러면 xclipboard에 현시된 내용이 지워 지고 계수기는 2로 계수한다. 사용자는 이 령역안에 복사된 본문을 붙이기할수 있다. Next와 Prev단추를 리용해서 여러개의 림시기억기를 하나하나 선택할수 있다.

중요한것은 일단 오려둠판창문의 현시기에서 선택한 본문이 있으면 그 본문은 복사되였다고 생각하여야 한다. 가운데마우스단추를 리용하여 필요한 곳에 선택된 본문을 붙이기할수 있다. 붙이기가 안된다면 본문을 선택하시오. 사용자는 오려둠판의 내용을 지울수 있으며(Delete) 그것을 파일로 보관(Save)할수도 있다. Microsoft Windows에는 이러한 복사와 붙이기기능이 없다.

12.12 표준X의뢰기

MIT는 X를 창문과 자원관리에 관계되는 폰트와 도형편의프로그람들, 탁상화면부속품들로 묶어 져있는 50여개의 의뢰기들과 함께 제공한다. Linux는 더 많은 의뢰기들을 가지고 있다. 다음절에서는 표준X에서 쓸모 있는 X부분품들을 론의할것이다. X의뢰기들의 목록에 대해서 알려면 /usr/X11R6/bin등록부를, CDE를 리용한다면 /usr/dt/bin 등록부를 보면 된다. 지금 론의하려는것들은 그림 12-1에서 본적이 있을수 있다.

12.12.1 상사시계(xclock)

모든 X배포물들은 상사시계 xclock와 함께 동작한다. 이것은 다음의 지령행에서 보여 주는것처럼 쓸모 있는 선택항목을 가지고 있다.

```
xclock -chime &           반시간 간격으로 종을 울린다
xclock -digital &         문자형식으로 인쇄한다
xclock -update 5 &        초눈금을 보여 준다
```

이 선택항목외에도 상사시계구성요소들의 색을 설정하는 다른 선택항목들도 있다. 이 시계는 언제나 화면의 모서리에 자리 잡는다.

12.12.2 수산기(xcalc)

X는 또한 수산기인 xcalc의뢰기도 실행시킨다. 두가지 방식 즉 Texas Instruments TI-30형 혹은 Hewlett-Packard 10C방식으로 실행된다.

첫번째 방식은 인수를 쓰지 않고 xcalc를 리용함으로써 계산된다. 이 수산기는 역뽈스까방식(reverse polish notation)(다시 말하여 HP-10C방식)을 리용하게 해주는 -rpn지령을 제외하고 특별한것이 없다. 이 방식은 표준방식에서보다 건누르기량이 적다. 계산식 3 + 5 + 9 = 는 역뽈스까방식으로 3 [Enter] 5 [Enter] 9 +로 된다.

12.12.3 체계부하의 현시(xload)

xload는 평균체계부하의 기둥도표(histogram)를 현시한다. 이것은 여러대의 원격기계상에서 실행될수 있으며 부하자료의 기둥도표를 자기의 국부기계에 가질수 있는 좋은 프로그람이다. 그때 가장 낮은 부하률을 가진 기계상에서 다음프로그람을 실행할수 있는데 다른것들도 같다(실례로 처리능력). 이 지령은 -update와 -jumpscroll선택항목과 함께 리용된다.

```
xload -update 15 -jumpscroll 1 &
```

기정적으로 xload는 5초간격(Linux에서는 10초)으로 기계를 조사한다. -update선택항목은 이 시간을 15초로 설정한다. -jumpscroll선택항목은 기둥도표가 오른쪽끝으로 계속 도달할 때 도형을 1만큼 왼

쪽으로 옮겨 준다. 이것은 표시기의 원활한 흐름을 보장해 준다. 이 지령은 여러대의 기계상에서 자주 실행되기때문에 창문은 꼭대기에 주콤퓨터이름을 보여 준다.

12.12.4 창문제거(xkill)

어떤 창문은 제목띠에 있는 단추를 찰칵함으로써 혹은 차림표단추로부터 어떤 항목을 리용함으로써 제거될수 있다. 때때로 창문은 이에 대해서 응답을 못하는 경우가 있는데 이 경우에는 한개 창문 혹은 모든 창문을 없애기 위하여 xkill을 리용하여야 한다. 인수들이 리용되지 않으면 X는 다음의 통보문을 현시한다.

```
Select the window whose client you wish to kill with button 1....
```

사용자는 이때 창문에로 유표를 움직이고 가르쳐 준대로 정확히 수행해야 한다. 어떤 어미창문을 제거하면 그 새끼창문도 물론 제거될것이다. -all선택항목은 사용자가 뿌리창문을 세번(매 단추로 한번씩) 선택하게 하여 모든 뿌리준위창문들을 단번에 제거한다.

12.13 시동파일(.xinitrc)

대부분의 응용프로그람들과 마찬가지로 X는 시동시에 특별한 보고느끼기를 주기 위하여 전용화될수 있다. X를 호출하기 위하여 startx를 리용한다면 그것은 xinit를 호출는데 xinit는 언제나 home등록부의 .xinitrc파일을 참조한다. 일반적으로 모든 X는 탁상화면에 xtern창문을 배치한다. 그 창문으로부터 다른 X의뢰기들을 실행시켜야 할것이다.

기동할 때 일부 X의뢰기들과 함께 적재되는 자기의 탁상화면을 요구하면 .xinitrc파일안에 지령렬을 넣어야 한다. 이 지령렬은 모든 지령(마지막의것은 제외)이 배경안에 넣어 지는 쉘스크립트(C쉘은 아니다.)이다.

실례로 .xinitrc는 다음과 같은것을 보여 준다.

```
$ cat $HOME/.xinitrc
xrdb -load $HOME/.Xdefaults
mwm &
xclock -g 100*100-0+0 -bg tomato3 -fg white -update 1 -chime &
xterm -g -80-55 -sb -sl 300 -bg darkslategrey -fg ivory -ry -iconic &
xterm -g -0-0 -sb -sl 3000 -bg darkslategrey -fg ivory &
xbiff -g +0+0 &
netscape
```

사용자가 mwm의뢰기가 기동했는가를 확인한다. 기동하지 않았다면 창문을 움직이거나 크기조절을 할수 없을것이다. xbiff는 우편통보도구(그림 12-1)인데 우편함에 우편이 도착하면 기발이 설정되고 삑소리가 난다. netscape는 다음 2개의 장에서 보게 되는 중요한 의뢰기이다.

12.14 X의 자원

모든 X의뢰기들은 흘림띠나 단추 등 많은 객체(부분품)로 구성된다. 객체들의 클라스는 물론 이 객체들의 개별적인 실체와 관련된 속성(자원)들이 있다. 실례로 모든 응용프로그람단추는 전경색을 가지고 있는데 그것을 단추에 대한 **자원**(rescurce)이라고 한다. 어떤 응용프로그람의 매 객체에 대한 모든 자원은 복잡한 항목들과 함께 의뢰기를 실행하지 않기 위하여 전용화될수 있다.

우의 .xinitrc실례파일은 배경색과 전경색(darkslategrey 와 ivory색) 그리고 흘림띠를 가지고 실행되는 xterm을 보여 준다. xterm이 언제나 나타나게 하기 위하여 $HOME/.xinitrc파일안에 이러한 속성들을 설정할수 있다. X는 기동될 때 이 파일을 읽는다.이 파일안에는 다음의 내용이 있어야 한다.

```
XTerm*ScrollBar: True
XTerm*saveLines: 10000
XTerm*background: darkslategrey
XTerm*foreground: ivory
```

정의는 체계에 따라 좀 차이날수 있지만 원리는 같다. 자원정의는 의뢰기(클라스)이름으로 시작하는데 뒤에는 부분품계층(점들로 구별되는 련속적인 객체)이라든가 모든 객체를 대신하는 별표식이 붙는다. 어느 경우에나 자원속성이 뒤따르는데 :뒤에 속성값이 놓인다. 이 값들은 론리값(True 혹은 False), 문자렬(ivory) 혹은 수값(10000)으로 될수 있다.

일단 이런 식으로 자원을 설정하면 이 속성들로 언제나 창문을 표시하고 싶은 경우 xterm과 함께 -sb, -s1, -fg, -bg선택항목을 리용할 필요는 없다.

12.14.1 자원의 적재(xrdb)

대화상태에서 .Xdefaults를 변화시켰다면 완료할 필요는 없고 재시동하시오. 그 구성파일을 재읽도록 X에 신호를 주기 위하여 간단히 xrdb지령을 리용할수 있다.

```
xrdb $HOME/.Xdefaults
```

X는 이 자원들에 대한 기정값들(일부는 응용프로그람 그자체안에 지적된것들)과 함께 시작한다. 체계령역구성파일 /usr/lib/xll/app-defaults(위치는 변할수 있다.)안에 설정되는것도 있다. 이 등록부에는 매 구성가능한 의뢰기의 뒤에 아름이 지어 진 파일이 존재한다.

CDE는 /usr/dt/app-defaults등록부를 리용한다. 체계에 리용되는 언어에 따라 이 등록부의 보조등록부를 리용해야 한다. 미국식영어가 리용된다면 그 등록부는 en_US.UTF-8등록부이다.

사용자가 자기의 home등록부에 .Xdefaults를 가지고 있지 않다면 체계령력파일의 명세는 X가 기동될 때 적용할수 있다. 전용화된 .Xdefaults파일을 만들었다면 의뢰기들이 오른쪽에 있는 자원값으로 호출되도록 .xinitrc안에 xrdb명령을 넣어야 한다.

12.14.2 자원들의 재정의(-xrm선택항목)

.Xdefaults안에 넣은것들은 물론 체계령역의 기정값설정들까지도 무시해야 할 때가 있을수 있다. 어떤 응용프로그람의 특별한 실체는 좀 다른 설정을 요구한다. 때때로 적당한 선택항목을 가진 지령을 호출함으로써 이 설정들을 재정의(override)할수 있다. 그러나 모든 자원들은 해당한 선택항목에 대응한것을 가지고 있지 않지만 -xrm선택항목만은 사용자가 어떤 자원값을 지적할수 있게 한다.

실례로 -bg선택항목으로 xclock의 배경색을 변화시킬수 있다. 그러나 그 클라스에 대한 자원명세를 리용할수도 있다.

```
xclock -xrm 'xclock*background: lightblue' &
```

이 간단한 자원들외에 사건을 해석하는데 관계되는 다른 설정들도 있다. 대표적인 사건은 마우스의 찰칵이나 움직임에 대한 해석이다. 그것들은 그리 직관적이지 못하다. 이에 대한 론의는 후에 하기로 한다.

이 장에서 본것들은 인터네트와 WWW에 자체로 접근하기 위하여 X에 대해서 알아야 할것들이다. 다음 2개의 장에서는 인터네트와 우편처리도구로서 Netscape를 광범히 리용할것이다.

요 약

X Window는 매점을 개별적으로 조작하는 비트배렬현시장치를 리용한다. 전체적인 현시장치를 뿌리창문이라고 하며 개별적인 응용프로그람들은 이 뿌리창문우에 창문들로 현시된다.

X는 startx와 xinit지령으로 시작된다. X는 xdm지령과 함께 체계를 기동하는 동안에 호출될수 있다. 그렇지 않는 경우 xdm은 뿌리사용자에 의해서만 실행될수 있다.

X는 두개의 구성요소 즉 의뢰기와 봉사기로 응용프그람을 분할한다. 봉사기프로그람은 감시기, 건반, 마우스를 조종하며 응용프로그람 그자체는 의뢰기이다. X프로그람들은 창문그리기에 대한 걱정이 없으므로 아주 간편하다.

X는 또한 TCP/IP망에서 실행되는데 의뢰기가 하나의 기계우에서 실행되면서 또 다른 기계의 현시장치를 가지는것도 십분 가능하다. xhost의뢰기는 봉사기에 접근하는것을 조종하는데 환경변수 DISPLAY는 의뢰기가 자기의 출력을 어디에 현시하겠는가를 규정한다. -display선택항목은 프로그람출력방향을 결정 짓기 위하여 어떤 X의뢰기와 함께 리용된다.

모든 창문들은 단추를 가지고 있는 제목띠를 가지고 있다. 왼쪽단추는 창문차림표를 펼쳐 준다. 창문은 이 세 단추들을 리용하여 움직이고 크기가 변경되고 최대화, 최소화될수 있으며 닫겨 질수 있다. 창문은 그우에 혹은 제목띠에서 찰칵됨으로써 초점을 가지게 될수 있다.

X의 창문들은 많은 부분품(widgets)들을 포함하고 있다. 그것들은 단추, 흘림띠, 검사칸, 단일선택단추, 표쪽, 내려펼침칸일수 있다. X의뢰기는 기정적으로 창문관리기능을 가지고 있지 않다. 특별한 의뢰기인 창문관리기는 X대화가 시작한 상태에서 곧 호출되여야 한다. 창문관리기는 창문을 움직이고 크기조절을 가능하게 만든다. UNIX의 표준창문관리기는 Motif(mwm)였는데 지금은 CDE의 dtwm으로 교체되였다. fvwm과 kwm은 Linux에서 가능한 창문관리기이다.

공통탁상환경(CDE)은 탁상화면과 창문관리기에 대해서 표준보기를 제공한다. CDE는 많은 응용프로그람을 시작할수 있는 정면판을 제공해 준다. 파일관리기는 파일과 등록부들을 조종하는 기능을 가지고 있는데 휴지통에 지워 진 파일들을 보관한다. 정면판은 또한 의뢰기들이 화면에 꽉 들어 차지 않게 하기 위하여 가상탁상화면을 만들게 해주는 작업공간스위치를 제공한다.

xterm창문안에서 UNIX지령행으로부터 모든 UNIX지령들과 X프로그람들을 실행시킬수 있다.

xterm은 흘림띠(-sb)와 함께 리용될수 있는데 보관될 지령행의 수도 지적될수 있다.

X프로그람들은 수많은 공통선택항목과 함께 실행된다. 그 선택항목들로 창문의 위치와 크기(-geometry), 전경색과 배경색(-fg)을 지적할수 있다. 그리고 아이콘으로 프로그람을 실행시킬수 있으며(-icon) 이름(-name) 혹은 제목(-title)을 제공할수 있다. 어떤 창문으로부터 마우스단추로 본문을 선택하여 복사할수 있다. 복사된 본문은 가운데단추를 찰칵함으로써 붙일수 있다. 복사되는 본문이 여러개일 때는 xclipboard의뢰기안에 기억될수 있다.

X에는 쓸모 있는 의뢰기들이 여러개 있다. X는 시계(xclock)와 수산기(xcalc)를 제공한다. xload는 체계부하상태를 현시하는데 원격기계들과 함께 자주 리용된다. xkill은 창문을 제거한다. 그것은 한번에 모든 뿌리준위창문을 제거할수 있다(-all).

X는 쉽게 전용화될수 있다. X의뢰기들은 xinit에 리용되는 시동파일 .xinitrc로부터 시작될수 있다. X자원들은 사용자가 어떤 X의 기능을 실지로 변화시키게 해준다. 이 특징들은 .xdefaults에 보관될수 있는데 xrdb는 이 파일을 읽을 때 리용될수 있다. 이 설정들은 모든 X의뢰기들에 가능한 -xrm선택항목으로 재정의될수 있다.

시험문제

1. 왜 X는 많은 비데오기억기를 요구하는가?
2. X를 기동하는데 리용될수 있는 두가지 지령의 이름을 말하시오. X를 시작하기 위하여 xdm을 리용할수 있는가?
3. 뿌리차림표를 어떻게 호출하는가?
4. X의 어느 부분품이 화면우에 창문을 현시하기 위한 책임을 가지고 있는가?
5. Solaris기계에서 실행되고 있는 xterm과 마찬가지로 X의뢰기들도 자기의 출력을 HP-UX기계에 현시할수 있는가?
6. xhost지령의 기능은 무엇인가? xhost +는 무엇을 의미하는가?
7. 선택된 창문을 어떻게 식별하는가?
8. 창문을 선택하지 않고 움직일수 있는가?
9. 검사칸과 단일선택단추와의 차이점은 무엇인가?
10. 창문차림표를 어떻게 호출하는가?
11. mwm이란 무엇인가?
12. 흘림띠와 함께 1000개의 행을 보관하면서 xterm을 호출하려면 어떻게 해야 하는가?
13. xinit지령이 탁상화면에 xterm창문을 놓지 않으면 그렇게 하기 위하여 그것을 어떻게 구성하겠는가?
14. $HOME/.Xdefaults 파일안에는 어떤 내용을 기입하는가?

련습문제

1. 왜 문자방식의 말단에서 도형을 현시할수 없는가?
2. rm으로 파일을 제거하는것과 휴지통으로 파일을 끌기하는 점과의 차이는 무엇인가?
3. X는 서로 다른 특징을 가진 여러가지 현시장치상에서 같은 프로그람을 실행시키는데서 제기되는 문제를 어떻게 해결하는가?
4. X현시장치를 구성하는 요소는 무엇인가?
5. 자기의 기계 saturn상에서 netscape를 실행시키고 romeo가 원격기계 uranus상에 있는 juliet의 현시장치에 이 출력을 현시하기 위하여 어떻게 할수 있는가?
6. 주콤퓨터 venus로부터 다음의 지령이 실행될 때 잘못된것은 무엇인가?

```
rsh saturn xload &
```

7. DISPLAY변수가 -display선택항목을 리용하는것보다 어떻게 더 편안한가?
8. 원격기계 saturn은 telnet기능을 가지지 못한다. 모든 의뢰기들과 함께 -display 선택항목을 사용하지 않고 venus로부터 원격적으로 프로그람을 실행시킬수 있는가?
9. 어느것이 창문우에 틀, 단추, 경계선들을 배치하는가? 봉사기인가, 의뢰기인가?
10. CDE에서 어느 프로그람이 창문관리기를 대신하는가?
11. 자기체계에 창문관리기가 없다면 어떤 현상이 일어 나겠는가? 사용자는 그래도 작업할수 있는가?
12. 탁상화면우에 5개의 X의뢰기들을 가지고 있다면 창문관리기를 실행시키는데 얼마만한 시간이 필요한가?
13. 그림 12-4의 중심에서 정면판우에 보이는 4개의 직4각형들의 기능은 무엇인가?
14. xterm은 어느 쉘을 리용하는가?
15. 제목띠에 다른 이름을 보여 주는 xterm창문을 어떻게 만들수 있는가?
16. 다음의 두 지령사이에서 본질적인 차이점은 무엇인가?

```
xterm -g 40x14+0+0
xclock -g 40x14+0+0
```

17. 왜 배경에서 X의뢰기들을 실행하여야 하는가?
18. 한 창문에서 다른 창문에로 본문을 어떻게 복사할수 있는가?
19. xterm창문우에서 실행되는 vi편집기로 본문을 복사한다면 사용자가 해야 할 추가적인 단계는 무엇인가?
20. 어떤 창문으로부터 여러 단락의 본문을 어떻게 복사하는가?
21. X를 완료하려면 xkill을 어떻게 리용할수 있는가
22. .xinitrc안의 모든 지령이 배경에 놓이게 된다면 어떤 현상이 일어 나겠는가?
23. 의뢰기를 실행시킬 때 자원설정을 어떻게 재정의할수 있는가?

제 13 장. 전자우편

전자우편은 자료통신매체로서의 팍스를 대신하는데 체신소의 부담을 덜어 준다. UNIX체계들은 아주 중요한 전자우편합신도구묶음을 제공한다. 이 도구의 사용자들은 TCP/IP망의 같은 주콤퓨터나 다른 주콤퓨터상에 있을수 있고 인터네트의 어떤 주콤퓨터에 있을수도 있다. 이러한 기능들로 하여 일부 사람들에게 있어서 UNIX는 전자우편을 의미한다.

전자우편매체는 비공식적이고 빠르며 값이 매우 눅다. 오늘 그것은 도형, 비데오, 음성파일도 조종할수 있다. 전자우편통보문을 보내려면 그안에 간단히 주소를 타자해야 한다(주소책으로부터 그것을 선택할수도 있다). 그다음 지령 혹은 단추를 찰칵하면 보내진다. 우표를 붙이는것도 봉투를 봉인하고 체신소에 보내는것도 없다. 통보문은 보통 몇초어간에 때로는 몇분내에, 몇시간내에 접수된다. 이로부터 전자우편사용자들은 체신소의 우편물을 《달팽이우편》이라고 부른다.

이 장에서는 3가지 문자방식의 우편처리프로그람 mail, elm, pine에 대해서 본다. 또한 X Window 체계에서 실행되는 아주 일반적이며 강력한 의뢰기인 Netscape Messenger를 론의한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 우편폴더의 구조와 우편통보문의 형식을 리해한다(13.1).
- mail지령으로 작업하기 위한 지식을 배운다(13.2).
- 우편물을 처리하는 elm과 같은 차림표방식의 프로그람을 리용한다(13.3).
- 차림표방식의 프로그람인 pine의 우월한 기능을 리용한다(13.4).
- .signature와 .forward파일의 의미를 파악한다(13.5).
- 우편물전송에서 SMTP와 POP통신규약의 역할을 리해한다(13.6).
- 우월한 우편물의뢰기로서 Netscape Messenger를 구성하고 리용한다(13.7).
- 멀리 떨어 져 있을 때 자동우편발송체계로서 vacation지령을 리용한다(13.8).
- 2진파일을 우편첨부물로서 전송하는데서 MIME의 역할을 배운다(13.9).

13.1 전자우편의 기초

어떤 주소화형식을 가지고 어떤 사용자에게로 오는 우편물은 그 사용자가 가입되지 않았다 해도 그 사람의 우편함에 넣어 진다. 이것은 talk지령에 비해 유리한 점을 가지고 있는데 talk에서는 두 사용자가 통신하려면 가입되여 있어야 한다. 우편형식으로 된 통보문은 말단에 직접 나타나지 않는다. 프로그람이 실행되고 있다면 체계는 다음의 통보문을 현시하기전에 프로그람실행이 완료되기를 기다린다.

```
You have new mail in /var/mail/romeo            romeo가 사용자이다
```

어떤 사람이 가입된 상태에서 아직 우편물을 보지 않았다면 체계는 우의 통보문을 가지고 그 사람에게 인사한다. 그것을 당장 처리할 필요는 없다. 즉 우편은 그 사람이 볼 때까지 우편함안에 남아 있다. 그러나 그것을 후에 보기로 하고 그 우편함안에 그냥 두기보다 도착하는 순간에 보내진 우편을 보는것이 실천적으로 좋다. 자기 말단에서 우편물을 본 다음 아래와 같은것들을 할수 있다.

- 그 송신자와 모든 수신자에게 응답한다.
- 그것을 다른 곳으로 보낸다.
- 그것을 우편함폴더에 보관한다.
- 개별적인 파일로 보관한다.
- 그것을 지운다.
- 그것을 인쇄한다.
- 주소책에 그 송신자와 모든 수신자들의 주소를 추가한다.
- 본문형식의 설명이 없다면 그것을 보기 위하여 방조응용프로그람을 호출한다.

우편에는 송신자가 보낸 날자와 시간이 표식되여 있다. 들어 오는 모든 우편은 우편함안에 보관된다. 이것은 보통 /var/mail(Linux에서는 /var/spool/mail)등록부안에 있는 본문파일인데 그 이름을 가입이름으로 가지고 있다. 모든 우편은 파일로서 우편함에 추가된다. charlie의 우편은 /var/mail/charlie안에 보관된다.

주해

이전의 UNIX체계들은 완충등록부(spool directory)로서 /usr/spool/mail을 리용한다. 쉘의 MAIL변수값을 평가함으로써 우편함의 위치를 찾을수 있다.

```
$ echo $MAIL
/var/mail/sumit
```

그러나 이 변수의 값을 변화시킨다고 하여 우편함의 위치가 변하는것은 아니다. 그것은 mail, elm과 같은 우편처리기들은 바로 우편함의 위치를 알기 위하여 이 변수가 정의된 파일을 읽는다.

우편통보문의 내용(머리부가 없다.)을 mail, elm 혹은 pine과 같은 문자방식의 우편처리기로 보았다면 이 통보문은 $HOME/mbox로부터 지워 질것이다. 이 우편처리기들은 통보문을 보관하고 보내고 안보내고 하는데서 서로 다른 폴더(folder)를 리용하는데 그것들은 $HOME/Mail안에 위치한다. 한편 Netscape가 $HOME/nsmail안에 있는 모든 폴더들을 보관한다.

총괄적으로 말해서 접수된 우편통보문을 찾기 위한 령역(area)이 있어야 한다.

- /var/mail (Linux에서는 /var/spool/mail): 우편은 후에 사용자ID로 이름 지어 진 어떤 파일안에 보관된다. 바로 여기서부터 통보문을 다른 곳으로 옮긴다. 전화회선을 통해서 우편을 되찾고 있다면 이 등록부에 접근하여야 한다.
- $HOME/mbox: mail과 pin은 이 파일로 우편을 옮긴다. 그렇지만 다른 우편처리기들은 자기의 등록부에 옮기지 않는다.
- $HOME/Mail/received: elm과 pine은 흔히 우편을 이 파일안에 보관하겠는가를 질문한다.
- $HOME/nsmail/Inbox: Netscape와 배타적으로 리용된다.

등록부위치들과 파일이름들은 우편처리기들을 구성함으로써 변화될수 있는데 그것들을 혼돈하지 말아야 한다. 그러나 이 모든 우편프로그람들은 들어 오는 우편들에 대해서 /var/mail안의 우편함파일을 본다.

13.1.1 주소지정도식

UNIX는 몇마일 떨어 진 여러명의 사용자들도 중앙기계에 가입해서 통보문들을 보내는데 리용할수 있는 시분할체계이다. 이전의 전자우편주소지정도식은 간단히 수신자의 사용자이름(/etc/passwd안에 등록된것)을 리용하였다. mail지령은 다음과 같은 식으로 리용된다.

```
mail henry              henry는 같은 주콤퓨터상에 있다
```

후에 기계들이 어떤 망에 접속될 때 사용자이름과 주콤퓨터이름의 조합은 주소가 전체 망안에서 유일하도록 되여 있다. 이 주소지정도식은 사용자이름과 주콤퓨터이름을 구분하는데 @를 리용한다. 다음의 지령은 다른 주콤퓨터 calcs에 있는 등록자리를 가지고 henry에게 우편주소를 지정하는 방법이다.

```
mail henry@calcs              henry는 주콤퓨터 calc상에 있다
```

이것은 우리가 talk(11.2)를 리용하던 방법과 같다. 인터네트의 령역이름체계(11.1)는 주소지정도식을 더 확장하였다. 주소는 세계적으로 유일해야 한다는것으로부터 주콤퓨터들은 령역(domain)과 련관된다. 실례로 령역 planets.com에서 주콤퓨터 calcs는 calcs.planets.com으로 어떤 전자우편응용프로그람에 알려질것이며 사용자 henry는 이 주콤퓨터에 대해서 henry @ calcs.planets.com으로 주소화될수 있다. 이 조합은 인터네트상에서 유일하다. henry가 접수하는 통보문에서 from: 과 To:행은 다음과 같다.

```
From: charlie@mumcs.net              다른 령역으로부터 오는 우편
To: henry@calcs.planets.com
```

이 이름짓기규칙은 전 세계의 모든 주콤퓨터들에 적용되였는데 전자우편은 인터네트상에서 가장 대중적인 응용프로그람으로 되였다(Web가 출현하기전까지).

13.1.2 우편통보문의 내부구조

모든 우편통보문은 여러개의 행으로 된 머리부정보로 구성되여 있다. 그 응용프로그람이 통보문을 만드는가 아니면 그것을 조종하는가에 따라 머리부들을 모두 볼수도 있고 보지 못할수도 있다. 그러나 대체로 통보문은 적어도 아래에서 보여 주는것처럼 처음에 4개의 마당을 보여 준다.

```
Subject: creating animations in macromedia director from GIF89a images
   Date: Mon,08 Nov 1999 15:42:38 +0530
   From: joe winter <winterj@sasol.com>
     To: heinz@xs4all.nl
     Cc: psaha@earthlink.net
```

우편통보문을 작성할 때 주제(subject)는 선택적이다. 그러나 주제마당을 꽉 채우는것이 좋은 방법인데 특별히 수신자의 주의를 끌어야 할 때 좋다.

Date:와 From:마당은 체계에 의해 자동적으로 채워 진다. 통보문을 보낼 때 우편에 잘못된것이 하나라도 있으면 되돌아 오는것처럼 수신자의 주소를 조심히 입력해야 한다.

다른 사람들에게도 통보문을 복사해서 보내고 싶을 때가 있다. 그러자면 carbon copy(Cc)항목에 주목하여야 한다. 같은 주콤퓨터상에 있는 다른 사람들에게 통보문을 여러개 복사해서 보낼 때 우편체계와는 달리 오직 한개 복사판이 우편처리기에 의해 보내진다. 복사판들은 목적지에서 만들어 지고 다음 그 사용자의 우편함에 전달된다.

주해

우의 실례에서 From:행은 송신자의 전체 이름을 보여 준다. 자기의 우편체계들이 인터네트용이라면 송신자에 대해서 최소형식에 의한것(<와 >안에 보여 준다.)보다 더 많은 정보를 제공하므로 이 형식을 리용해야 한다. 그러나 왼쪽문자렬(joewinter)은 통보문경로를 정하는데 리용되지 않으므로 오유가 발생해도 우편물을 되돌려 보내지 않을것이다. 그러나 사용자가 송신자에게 응답하는데 winterj 혹은 sasol.com이 정확하지 않게 씌여 졌다면 우편은 분명히 되돌아 올것이다. 이에 대해서는 13.7에서 본다.

13.2 전통적인 우편처리기(mail)

UNIX체계가 제공하는 초기의 우편처리도구는 mail프로그람이였다. 이 프로그람의 편리성과 단순성은 그 어떤 편집기능이 없이 파일을 만드는 cat의것(cat > foo)과 적당히 비교될수 있다. 그러나 mail은 누군가에게 한행 혹은 두행을 보낸다면 아주 편리한 도구이다. 사실 접수된 통보문을 처리하는것보다 mail로 통보문을 보내는것이 더 간단하다.

mail을 실행시키는 방법은 두가지가 있다. 수신자의 전자우편주소와 함께 그것을 리용하면 발송방식에서 그것을 리용하는것으로 된다. mail은 표준입력을 요구하는데 일부 체계에서는 [ctrl-d]로, 다른 체계에서는 .으로 완료한다.

```
$ mail henry              henry는 같은 주콤퓨터상에 있다
Subject: RealAudio - TCP or UDP?
Dear henry:
Which protocol does RealAudio use - TCP or UDP?
I think it's UDP
[ctrl-d]
$ _               Solaris는 점(.)을 사용한다
```

통보문을 보내는것은 이처럼 간단하다. 우편은 henry의 우편함안에 넣어 지는데 보통 /var/mail/henry 이다. 앞에서 본바와 같이 모든 전자우편프로그람들은 들어 온 우편에 대하여 이 파일을 읽는다.

우편통보문의 내용은 또한 표준입력을 절환함으로써 제공될수 있다. Subject:행은 입력이 다음과 같이 절환되면 빈 공간으로 된다.

```
mail henry < note.lst
```

이것이 바로 《머리부가 없는》 프로그람이 가지는 유리한 점이다. 여기서 mail은 비대화적으로 동작하므로 자동적으로 통보문들을 전송하기 위하여 쉘스크립트안에서 그것을 리용할수 있다. elm도 이것을 허락하지만 pine은 아니며 Netscape도 분명히 아니다.

우편을 보려고 henry는 mail프로그람을 호출할수 있는데 이때 아무런 인수도 없다.

```
$ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/henry": 1 message 1 new
>N  1 sumit@saturn  Sat Sep 25 17:34  15/455  "RealAudio - TCP or UDP?"
? _               우편프롬프트
```

처음에 mail도 통보문의 머리부를 보여 주는데 렬번호, 송신자의 전자우편주소(sumit@saturn), 날자와 시간, 행수와 바이트크기(15/455) 그리고 마지막으로 그 통보문의 머리부를 보여 준다. 그다음 ?프롬프트로서 사용자입력을 대기한다. mail은 이 프롬프트에서 리용될수 있는 몇개의 한문자내부지령을 가지고 있다. 여기서는 다만 그 일부만 론의한다.

1로 표식된 통보문을 보기 위하여 henry는 통보문번호를 입력하고 다음에 [Enter]를 입력해야 한다. henry는 w 1로 통보문을 파일로 보관할수 있으며 혹은 d 1로 그 파일을 지울수 있다. 마지막에 mail

을 완료하려면 q를 리용하여야 한다. 일단 통보문을 보면 /var/mail/henry로부터 지워 지고 $HOME/mbox 에 추가된다. 이 이동은 mail과 pine에서 진행되지만 elm에서는 아니다. mail프로그람의 작업세부를 더이상 설명하지 않겠다.

리용범위가 제한된 mail일지라도 그것은 기본적인것들을 발전시켜 더 좋은 우편도구로 되였다. UNIX체계들은 2개의 특별한 프로그람 elm과 pine을 제공하는데 리용하기 쉽다. 이 프로그람들은 완전히 차림표화되고 리용하기 쉬우며 아주 쓸모 있다.

13.3 화면지향우편처리기(elm)

elm이 지령이름으로 실행되면 전체 화면을 꽉 차지한다. 그때 그 화면의 밑에 내부지령들이 보일것이다. 이 종합적인 방식에서 통보문들을 보내고 접수하는데 elm을 리용할수 있다. elm은 기동한 상태에서 접수된 통보문의 머리부를 mail형식에 따라 적당히 수로 보여 준다. 대표적인 elm화면은 그림 13-1에 보여 준다.

```
Mailbos is `/var/mail/henry` with 3 messages [ELM 2.4ME+ PL37 (25)]

   1    Sep 24 Julie Brown        (55)   Going Out?
     2    Sep 25 system PRIVILEGED  (30)   Accounting time less than 50 hours
   N 3    Sep 24 Kothari Pioneer    (269)  KPMF InstaNAV 24.09.99

 ...blank lines removed...

     You can use any of the following commands by pressing the first character;
  d) elete or u) ndelete mail, m)ail a message, r)eply or f) orward mail, q)uit
        To read a message, press <return>. j = move down, k = move up, ? = help

  Command: _                                                Cursor waiting here
```

그림 13-1. elm화면

차림표는 아주 알기 쉬우며 초학도들에게도 별다른 문제는 없을것이다. 읽지 않은 새 통보문들은 왼쪽에 N으로 지적된다. 통보문의 크기는 괄호안에 바이트로 보여 준다. 어떤 통보문을 보려면 강조색을 움직여 통보문을 선택한 다음 [Enter]를 누른다.

```
Message 1/3 Julie Brown              1998년 4월 30일 오전 10:02:12 +0530
Return-Path: <local>
Date: Thu, 30 Apr 1998 10:02:12 +0530
To: henry@saturn
Subject: Going Out?

Dear Henry:
Let's eat out since Sonu doesn't need to go to school
For the next three days
```

기본차림표로부터 q를 선택하면 elm을 완료하는데 다음의 질문에 대답하기전에는 완료를 하지 못한다.

```
command: Quit                    Move real message to "received" folder? (y/n) y
command: Quit                    Keep unread message incoming mailbox? (y/n) y
```

우에서 첫번째 프롬프트에 y를 입력하는것은 읽은 통보문을 $HOME/Mail/receved파일에 추가하겠다는것을 의미한다. 두번째 프롬프트에 y를 입력하는것은 elm이 읽지 않은 통보문들을 /var/mail/henry에 보관하겠다는것을 의미한다. 기타 경우 elm은 이 통보문들을 접수한 같은 폴더에 옮긴다. elm은 mbox를 리용하지 않는다는것을 알아야 한다.

주해

같은 기계상에 있는 누군가에게 자기의것으로 주소를 지정하고 있다면 기계이름을 빼고 간단히 To:마당에서 사용자이름을 리용할수 있다.

13.3.1 통보문의 전송

우편통보문을 보내는것은 아주 간단하다. 처음에 기본차림표로부터 m을 선택하여야 하며 그다음 다음의 프롬프트에 응답해야 한다.

```
Send the message to: jackie@caltiger.com
Subject of message: Yet another proposal
Copies to: hillary@caltiger.com
```

그다음 elm은 통보문을 편집할수 있는 vi(혹은 emacs)편집기를 기동시켜 펼쳐 놓는다. 새로운 통보문을 보내겠는가 혹은 어떤 통보문에 응답하겠는가는 다른 사람에게 보내는 복사판들에 표식할수 있다. 편집기를 완료한후에도 여러개의 항목을 선택할수 있다.

```
Please choose one of the following options by parenthesized letter: s
        e)dit message, edit h)eaders, s)end it, or f)orget it.
```

통보문을 편집하고 취소하는것과 같은것은 일반적으로 할수 있는데 아주 중요한 항목 즉 통보문전송을 연기하는 기능이 여기서 빠졌다는것을 알아야 한다. 때때로 어떤 조건에서는 통보문을 보내고 싶지 않을 때가 있다. 그것이 바로 pine과 Netscape Messager가 elm과 차이나는 점이다.

13.3.2 접수한 통보문의 처리

통보문을 보내겠는가 지우겠는가에 응답할수 있다. 응답하려면 통보문을 선택하고 r를 누른다. 다른 곳으로 복사판들을 보내기 위한 선택항목도 가지고 있다.

```
Command: Reply to message                          Copy message? (y/n) n
Subject of message: Re: The forthcoming holidays
Copies to:
```

편집기가 다시 기동할것이다. 통보문을 작성하고 그다음 탈퇴하기전에 작업한 내용을 보관하시오. elm은 새로운 통보문을 보낼 때 보았던 꼭 같은 선택항목들을 마지막으로 제공한다.

f로 다른 사람에게 더 발송할수 있다. 또 d로 통보문을 지울수 있지만 elm은 q로 프로그람을 완료할 때 오른쪽의것을 하겠는가 질문한다.

```
Command: Quit                                      Delete message? (y/n) y
```

13.3.3 스팸의 처리

자기가 적극적인 전자우편사용자로 되면 우편목록에 등록하고 Web싸이트에 자기의 전자우편주소를 넘겨 주어야 할것이다. 나중에는 많은 사람들이 자기의 주소를 알게 될것이며 우편함에는 매일 생각지도 못한 통보문들이 쌓일것이다. 그러면 저도 모르게 **스팸우편**(spam mail : 불필요한 우편)들이 쓸어 들게 된다. elm을 포함한 대부분의 전자우편프로그람들은 우편속에 탐색문자렬을 식별하는 려파규칙과 그 문자렬을 포함하고 있는 모든 통보문들을 우편함에서 지우는 기능을 제공한다.

/지령(혹은 통보문내용을 탐색하려면 //)으로 탐색하려는 패턴을 지적할수 있다(vi형식에서). 일단 그 탐색이 보고 싶지 않은 통보문들을 정확히 식별하였다면 질문없이 그 통보문들을 [ctrl-d]로 지울수 있다.

13.3.4 elm의 선택항목

elm은 어떤 지적된 곳에 있는 우편폴더를 읽는데 리용될수 있다. 이때 -f선택항목의 리용이 필요하다. 이 선택항목을 리용하여 Netscape의 Inbox폴더를 읽을수 있다. 사용자는 Netscape가 $HOME/nsmail안에 있는 모든 우편파일들을 보관하도록 재호출할것이다.

```
elm -f $HOME/nsmail/Inbox
```

자기 우편함안에 우편이 있는가 없는가를 검사하려면 -z선택항목과 함께 elm을 리용한다

```
$ elm -z
You have no mail.              그 프로그람도 완료한다
```

elm은 인수로서 사용자이름과 함께 리용될수 있다. 본문은 또한 표준입력으로부터 주어 질수 있다.

```
elm -s "The forthcoming holidays" henry          -s는 Subject:행을 지정한다
elm henry <holiday.txt>                 표준입력으로부터의 통보문
```

elm의 동작은 $HOME/.elm/elmrc파일을 편집함으로써 진행될수 있다. 여기서 편집기를 선택하고 (editor = vi) 우편을 보관하는 등록부를 설정하며(maildir = $HOME/Mail) 송신자 혹은 수신자의 사용자ID로 통보문들을 보관할수(savename = on) 있다. 또한 elm의 o지령을 리용하여 일부 이러한 기능들을 설정할수도 있다. elm의 중요한 지령들은 표 13-1에서 보여 준다.

표 13-1. elm의 내부지령

지 령	의 미
m	통보문을 우편으로 보낸다
r	현재의 통보문에 응답한다
d	현재의 통보문을 지운다
f	현재의 통보문을 전송한다
q 혹은 Q	q로서 탈퇴 혹은 통보문을 지우거나 제거하기 위하여 Q로서 질문없이 탈퇴한다
s	등록부(기정적으로 $HOME/Mail)에 현재의 통보문 혹은 꼬리표가 붙은 통보문을 보존한다.
/	패턴에 대하여 From:와 Subject:을 탐색한다
//	패턴에 대하여 전체 통보문을 탐색한다
[Ctrl-d]	패턴을 포함하는 통보문을 지운다
o	elm선택항목창문을 연다

참고

기정구성으로 elm을 리용하려면 vi편집기를 알아야 한다. emacs나 pico와 같은 간단한 편집기를 리용하고 싶다면 o지령을 리용하여 elm의 구성을 변화시키고 E와 관계되는 행을 변화시키고 [Enter]를 누르며 그다음 그를 리용하여 새로운 구성을 보관한다.

주해

mail과 elm지령(pine도 포함)의 특징적인 기능은 그것들이 here문서(19.2)로 알려 진 특별한 방향절환기호(<<)와 함께 리용된다는것인데 그 입력은 꼭 같은 스크립트파일안에 배치된다. 이것은 쉘이 제공하는 아주 쓸모 있는 기능이다.

13.4 또 하나의 우편처리프로그람(pine)

인터네트새소식이나 전자우편을 위한 프로그람인 pine은 일반적으로 문자방식의 우편처리기인데 인터네트상에서 광범히 리용된다. elm과 같이 이것도 안내서들을 읽지 않고 리용할수 있다.

pine이라고 쓰면 그 우편처리기가 호출된다. 대면부는 아주 단순하며 실제로 위력하다는것을 느낄수 있다. pine의 열기화면은 그림 13-2에서 보여 준다.

```
PINE 4.05    MAIN MENU                      Folder : INBOX  5 Messages

        ?     HELP               -  Get help using Pine

        C     COMPOSE MESSAGE    -  Compose and send a message

        I     MESSAGE INDEX      -  view messages in current folder

        L     FOLDER LIST        -  Select a folder to view

        A     ADDRESS BOOK       -  Update address book

        S     SETUP              -  Configure Pine options

        Q     QUIT               -  Leave the pine program

 Copyright 1989-1998.  PINE is a trademark of the University of Washington.

                  [ Folder  “INBOX” opened with 5 messages]
? Help                      P PrevCmd                 R RelNotes
O OTHER CMDS L [ListFldrs]  N NextCmd                 K KBLock
```

그림 13-2. pine의 열기화면

차림표항목들은 가지런히 놓여 있다. 사용자는 주소책을 보유(elm에 없는 기능)할수 있다. 기본차림표가 일반적인 도움말기능을 제공해도 pine은 ?를 리용함으로써 호출되는 문맥의존도움말(context-sensitive help)도 제공한다. 그리고 다른 경우에는 ^G([Ctrl-g])가 리용된다.

13.4.1 우편보기

pine은 등록부안으로 우편이 들어 오고 나가고 하는것을 정지시킨다. elm도 그렇게 하지만 pine은 등록부의 목록도 현시할수 있다. l 혹은 L을 누르면 목록을 보여 주는데 그림 13-3에 실례를 보여 준다.

이 5개의 등록부(때로 그이상)중에 어느 하나를 강조할수 있는데 [Enter]를 눌러 세부적으로 볼수 있다. 이 지령(l 혹은 L)은 이미전에 보내거나 연기된 통보문들을 보게 해준다. INBOX폴더(Netscape에 리용되는 용어와 같다.)는 /var/mail 혹은 $HOME/mbox안에 있는 사용자의 실패파일을 참조한다. 이 등록부를 강조하고 [Enter](혹은 i)를 누르면 머리부들(통보문색인)이 현시된다(그림 13-4).

pine은 화면의 마지막 두 행에 문맥의존도움말을 현시한다. 어떤 화면으로부터 다른 화면으로 움직인다면 이 두 행에서 변화된 항목을 볼것이다.

elm에서처럼 유표건들로 통보문들을 선택하는데 통보문내용을 보려면 [Enter]혹은 >를 누른다. 이 폴더에서 읽지 않은 통보문(+N으로 표식된것)을 보기로 하자(그림 13-5).

```
                    Local folders in Mail/
        ------------------------------------------------------------
        INBOX      sent-mail      saved-messages     postponed-msgs
        received
```

그림 13-3. pine에서 등록부목록

```
PINE 4.05   MESSAGE INDEX                 Folder: INBOX Message 1 of 26 NEW

      1 Aug 23 isingh@elnet-31.com   (304) Mail server working fine
      2 Aug 22 sujit lala          (3,993) Fw: GOT IT
+ N   3 Aug 23 henry blofeld       (8,174) RE: RealAudio -- TCP or UDP?
      4 Aug 23 Kothari Pioneer    (15,561) KPMF InstaNAV 23.08.1999
      5 Aug 26 The Economist       (9,639) Politics This Week August 21st
...lines deleted...

? Help       < FldrList    P PrevMsg      - PrevPage  D Delete    R Reply
O OTHER CMDS > [ViewMsg]   N NextMsg    Spc NextPage  U Undelete  F Forward
```

그림 13-4. pine에서 통보문머리부창문

```
Date: Mon, 23 Aug 1999 13:12 +0500 (GMT)
From: henry blofeld <henry@vsnl.com
To: joe winter <winterj@sasol.com>
Subject: RE: RealAudio -- TCP or UDP?

dear joe:

RealAudio has to come streaming in. If it has to wait for
retransmission of segments, will you ever hear the sound properly?
It has to use UDP.
...lines deleted...

? Help       < FldrList   P PrevMsg      - PrevPage  D Delete    R Reply
O OTHER CMDS > [ViewMsg]  N NextMsg    Spc NextPage  U Undelete  F Forward
```

그림 13-5. pine에서 통보문내용창문

또 다른 통보문을 더 보려면 원래상태로 돌아 와서 n을 리용하여 볼수 있는데 이전 상태로 돌아 오려면 p를 리용한다. i를 누르면 통보문색인상태로 돌아 온다.

참고

>와 <건은 아주 쓸모 있으며 pine에서 작업을 할수 있게 한다. 반복적으로 >를 리용하면 폴더목록으로부터 머리부목록에로 옮길수 있는데 그다음 그 통보문내용을 특징 짓는 화면에로 옮겨 진다. 통보문이 어떤 첨부물들을 가지고 있다면 또 다른 >는 첨부물색인과 부속된 내용을 보여 줄것이다. X로부터 pine을 리용한다면 순서대로 실행된 련속적인 >들은(그사이에 방향건과 함께) 외부적인 도형보기프로그람을 빨리 호출할수 있으며 화상을 보여 줄수 있다. 이제까지의 경로를 되돌아 오려면 <를 리용한다. 즉 pine의 열기화면으로 곧바로 되돌아 온다.

13.4.2 통보문의 구성

c를 누르면(주차림표로부터) pine은 통보문을 작성하기 위하여 그안에 내장된 pico편집기를 불러 들인다. 이때 국부적인 사용자에게 통보문복사판을 보낸다고 하자. 그림 13-6은 구성창문을 보여 준다.

```
To      : bruno@elnet-31.com                인터네트상의 어떤 사용자
Cc      : henry                             같은 주콤퓨터상의 사용자
Attchmnt:
Subject : how does one start Star Office?
---- 통보문본문 ----
dear bruno:

I am unable to invoke Star Office from an ordinary user user account. well,
henry is using some word processor on linux (I don't know which one), and
he may be able to help too. do either of u know?

sumit
...lines deleted...

^G Get Help  ^X Send     ^R Read File ^Y Prev Pg ^K Cut Text  ^O Postpone
^C Cancel    ^J Justify  ^W Where is  ^V Next Pg ^U UnCut Text^T To Spell
```

그림 13-6. pine의 구성창문

이 편집기에서는 elm에서 쉽게 할수 없는것들까지도 하기 쉽다. 본문을 자르고(^K) 붙일수(^U) 있으며 문자렬을 탐색할수(^W)도 있다. 본문구성을 끝낸후 그것을 보내기 위하여 ^X를 리용할수 있다. pine은 후에 취소할수 없는 모든 동작을 사용자가 확인하는 원칙에서 설계되였다.

```
Send message?
```

pine은 또 다른 점에서 elm과 차이나는데 그것은 사용자가 통보문전송을 연기(^O)하게 하는것이다. 이것은 통보문을 보내겠다고 결심하기전에 통보문을 다시 편집하게 하는 쓸모 있는 기능(Netscape에서도 가능하다)이다. 맞춤법검사프로그람을 실행시킬수 있는데(^T) 이것은 일반적으로 체계와 함께 동작하는 프로그람인 spell 혹은 ispell을 리용한다. 이 프로그람은 $HOME/.pinerc안에서 speller변수의 설정에 따라 조종된다. Linux사용자들은 /usr/bin/ispell에 이 변수가 명백히 설정되였는가를 확인해야 한다.

주해

그림 13-6에 보이는 조종건들은 유표가 화면의 내용안에 위치하고 있을 때만 보인다. 유표가 머리부에 있다면 여러가지 건묶음에 제시된다.

내용부분에 유표가 위치하고 있을 때 통보문안에 어떤 파일의 내용 즉 프로그람목록이나 그것의 출력을 삽입하려면 ^R를 리용할수 있다. 파일을 첨부물로 보내려면 머리부부분(Attchmnt:)에 유표를 옮기고 ^J를 리용한다.

13.4.3 발송과 응답

발송과 응답기능들은 elm과 류사하므로 여기서 상세하게 서술할 필요는 없다. 통보문색인창문이나 통보문내용을 보기할 때 이 기능을 수행하기 위하여 f와 r를 리용한다. pine은 응답하기로 결심하였을 때 질문을 제기한다.

```
Include original message in Reply?
Use "Reply-To:" address instead of "From:" address?
Reply to all recipients?
```

Reply-To:프롬프트는 송신자의 통보문이 이 마당을 포함할 때 나타난다. 송신자의 기계는 가동하지 않을수 있는데 그때 그는 통보문을 우편으로 보내기 위하여 자기 친구의 기계를 리용하고 있을수 있다. 리해하기 쉽게 그는 친구의 주소로가 아니라 자기의 실제주소로 응답하고 싶을것이다. 이 기능은 elm과 pine으로 통보문을 작성하였을 때에는 불가능하다. 그래서 Reply-To:마당과 함께 통보문을 보려면 통보문는 또 다른 우편처리기(Netscape)로 작성하여야 한다.

회답속에 본래의 통보문을 그대로 포함시킬수 있다. 이것은 통보문으로 호상련계를 가지는데서 수신자를 자주 도와 주는데 사람들은 자기들이 작성한것을 잊어 먹는 경향이 있다. 또한 사용자자신이 개별적으로 모든 수신자들에게 회답할 필요가 없게 하며 우편목록(14.3)을 훑어 보는 기능이 있는데 인터네트상에서 광범히 리용된다.

13.4.4 주소책

pine의 흥미 있는 기능은 **주소책**(Netscape에서도 가능)인데 일단 여기에 어떤 사람의 세부를 입력하면 후에 이 《책》으로부터 주소를 얻을수 있다. 주차림표로부터 주소책을 선택하고 그다음 새로운 항목을 추가하기 위하여 @를 입력할수 있다.

```
NickName  : chris
Fullname  : christophe flynn
Fcc       :
Comment   :
Addresses : chris@lakeerie.edu

Fill in the fields just like you would in the composer.
To form a list, just enter multiple comma-separated addresses.
It is ok to leave fields blank. Press "^X" to save the entry, "^C" to cancel.
If you want to use quotation marks inside the Fullname field, it is best to use
single quotation marks; for example: George 'Husky' Washington.
```

통보문이 pine으로부터 나타난다. ^X로 내용을 보관한후 통보문을 작성할 때 주소를 꺼낼수 있다. 첫번째 프롬프트(To:)에서 주소책을 호출하려면 ^T를 누른다. 그다음 원하는 내용을 강조한 다음 s로 그것을 선택한다. 그 마당은 자동적으로 채워 진다.

13.4.5 pine의 선택항목과 구성

pine은 많은 선택항목들을 가지고 있다. -I선택항목은 pine이 기동할 때 실행되여야 할 동작들을 건누르기로 제공하게 한다. 구성창문을 열기 위하여 주차림표에서 c를 누르면 pine이 기동할 때 이 창문이 제시된다.

```
pine -I c              기동한 상태에서 c지령을 실행시킨다
```

또한 주소와 함께 직접 pine을 리용할수도 있다.

```
pine henry@saturn              구성창문이 나타난다
```

elm이 $HOME/Mail의 received폴더안에 기정으로 보관된 통보문들을 볼수 있다.

```
pine -f received
```

pine은 주차림표로부터 기동될수 있다. 그것은 또한 체계령역설정을 위해서는 시동파일 /usr/lib/pine.conf들을 리용하고 재정의될수 없는 선택항목들에 대해서는 usr/lib/pine.conf.fixed를 리용한다. pine은 또한 사용자고유의 파일 $HOME/.pinerc를 제공한다. 이 파일은 문서로 되여 있는데 그것들을 편집할 필요는 없을것이다.

elm과 pine은 적어도 하나의 원인으로 인터네트상에서 리용된다. 즉 그것들은 둘 다 다매체첨부물(13.9)들을 보내는데 리용될수 있기때문이다. WWW의 출현으로 elm과 pine의 인기는 내리막길에 들어섰다(Netscape의 Communicator 쏘프트웨어도 우편을 처리하기때문에). Netscape는 이 장에서 론의해야 할 마지막우편처리프로그람이다.

주해

-attach선택항목을 리용하는 pine으로 본문파일이나 2진파일을 첨부물로서 보낼수 있다. elm에서는 -A선택항목으로 그와 같은 기능을 수행한다. 그러나 보내지는 파일의 형은 이 우편처리기들이 리용하는 어떤 중요한 구성파일안에 지정되여야 한다. 다매체첨부물을 처리하려면 먼저 MIME를 리해하여야 하는데 마지막절에서 론의된다.

13.5 두가지 중요한 파일들(.signature와 .forward)

대부분의 우편처리기들은 보내려는 모든 통보문들에 어떤 단락을 추가해 줌으로써 리용하기 편리하게 해준다. 우리는 앞에서 모든 통보문의 끝에서 어떤 사람의 주소와 전화번호세부들을 본적이 있을것이다. 대체로 송신자는 통보문을 보낼 때마다 이러한 세부들을 자체로 입력하지 않고 자동적으로 파일내용을 붙이는 우편처리기 즉 .signature를 리용한다.

이와 같은 동작을 하려면 자기의 홈등록부안에 있는 .signature에 모든 세부들을 넣어야 한다. elm과 pine은 기정으로 이 파일들을 리용한다. Netscape도 모든 세부에 대한 구성파일을 가지고 있으면서 이 파일을 리용한다. 이러한 모든 우편처리기들은 다른 파일을 리용하도록 구성될수 있다.

다른 파일인 $HOME/.forward는 자동적으로 우편을 발송하는 기능을 제공한다. 어떤 사람이 려행을 할 때 그 사람은 자기의 우편을 다른 주콤퓨터 다시말해서 Web상에 제공된 전자우편싸이트로 보내고 싶을수 있다. 그러면 .forward파일에 다음과 같은 내용을 기입해야 한다.

```
romeo@yahoo.com
```

이것은 모두 필요되는것들이다. 주소가 romeo로 되여 있는 통보문을 우편봉사기가 접수하였다면

romeo의 우편함에로 통보문을 보내지 않고 Yahoo의 싸이트에로 통보문을 보낸다. .forward를 통해서 발송하는것이 sendmail(SMTP를 리용한면서 인터네트상에서 대부분의 우편을 전달하는 프로그람)의 한 가지 기능이기때문에 우편처리기가 이 기능을 리용하도록 구성될 필요는 없다.

다른 주콤퓨터에 .forward로 우편을 발송하고 그다음 회송하는 기능을 설정하는 경우에는 문제가 발생한다. 이런 상태는 .forward에 의존하는 류동사용자(mobile user)들에게서 발생할수 있는데 forward는 류동사용자들이 어떤 통보문을 놓치지 않았는가를 확인한다. 량쪽에서 발송하면 순환이 형성되며 통보문은 "착륙"기회를 얻지 못한다. 통보문은 sendmail이 개입하여 고리를 끊어 주기전까지는 계속 앞뒤로 왕복한다.

참고

elm이 .signature를 리용하지 않으면 $HOME/.elm/elmrc파일을 편집하고 국부우편에 리용하겠는가 원격우편에 리용하겠는가에 따라 localsignature 혹은 remotesignature를 $HOME/.signature에 설정한다. pine이 .signature를 무시하고 있다면 $HOME/.pinerc안에 signature-file을 설정한다. Netscape는 Edit>Preferences>Mail & Newsgroups>Identity에 설정을 하게 한다.

13.6 우편은 어떻게 동작하는가

우리는 Netscape를 론하기전에 우편이 동작하는 방법을 알아야 한다. elm이나 pine과 같은 문자방식의 의뢰기들로 고정시키려 한다면 이 부분을 뛰여 넘을수 있다. 그러나 Netscape의 열렬한 사용자라면 그리고 Netscape를 구성하려는 사람이라면 이 부분은 무조건 읽어야 한다.

telnet나 ftp(단순한 의뢰기-봉사기환경안에서 작업하는)와는 달리 우편처리는 3가지대행체(agency)들의 작업을 요구한다.

- **우편사용자대행체**(MUA:mail user agent): 우편을 읽고 보내기 위한 작업
- **우편전송대행체**(MTA:mail transport agent): 기계들사이에 우편을 발송하기 위한 작업
- **우편배포대행체**(MDA:mail delivery agent): 수신자의 우편함에 우편을 배포하기 위한 작업

이 3개 층으로 된 배렬에서 pine이나 elm과 같은 우편사용자대행체(MUA)는 사용자의 앞단(front-end)으로서 동작한다. 그러나 그자체로서 망을 통하여 우편을 전송하도록 설계되지는 않는다. pine과 Netscape와 같은 일부 MUA들은 그렇게 전송할수 있는 능력도 가지고 있지만 UNIX체계들은 일감에 따라 배당된 개별적인 대행체를 가지고 있다.

우편전송대행체(MTA)는 두가지 기능 즉 우편을 보내고 접수하는 기능을 가지고 있다. 보내는 쪽의 MTA는 수신자의 주소를 확인하며 수신자쪽에 있는 MTA에 직접 통보문을 배포한다. 우편의 보내기와 접수하기는 일반적으로 **단순우편전송규약**(Simple Mail Transfer Protocol: SMTP)에 의해 조종된다.

MTA는 통보문의 최종배포에 응답하지 않는다. 접수하고 있는 MTA로부터 우편을 받아서 실제사용자의 우편함에 배포하는것이 바로 우편배포대행체(MDA)이다. 이것은 /bin/mail(이것도 역시 MDA이다.)과 deliver(procmail은 Linux의 표준 MDA이다.)와 같은 개별적인 프로그람에 의해서 조종된다.

4번째 층은 MTA가 MDA에로 우편을 넘겨 줄수 없을 때에 쓰인다. 이것은 접수하고 있는 주콤퓨터가 망에 접속하는 전화회선에 대해서 성립한다. 이 배렬에서 사용자들은 대체로 우편이 들어 오고 나가는것을 조종하는 인터네트봉사제공자(Internet Service Providers:ISP)의 기능을 리용한다. ISP는 봉사기상에서 사용자의 우편을 보관한다. 그리고 사용자는 개별적인 프로그람을 리용하여 우편을 가져 온다. 오늘날 우편을 가져오는데 리용하는 통신규약으로서는 우편국규약(Post Office Protocol:POP3)과 인터

네트통보문접근규약(Internet Message Access Protocol:IMAP)이 있다.

그림 13-7은 SMTP와 POP가 하나의 망안에서 우편을 움직이는 방법을 보여 준다. Netscape는 많은 우편처리기능들을 내장하고 있으며 체계가 제공하는 일부 우편처리봉사들을 무시하도록 구성된다.

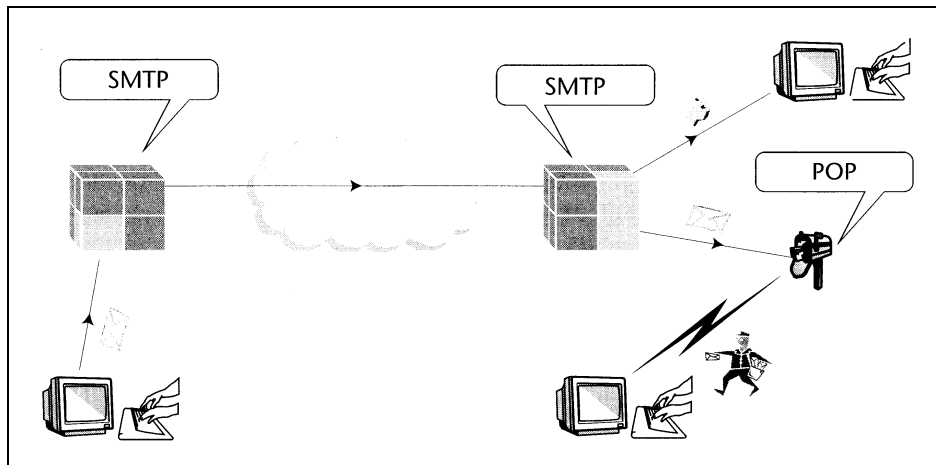



그림 13-7. SMTP와 POP에 의한 우편의 이동

13.7 가장 강력한 우편처리기(Netscape Messenger)

Netscape Messenger는 의뢰기로 실행되는 훌륭한 MUA이다. 사실 Messenger는 Navigator를 특징 짓는 Netscape Communicator의 구성요소이다. Messenger를 호출하려면 xterm창문에서 다음의 두가지 방법중의 한가지로 netscape지령을 실행시킨다.

```
netscape &
netscape -messenger &                Messenger만 호출된다
```

첫번째 지령은 일반적으로 Messenger와 함께 혹은 Messenger가 없이(이것은 Communicator가 구성되는 방식에 의존된다.) 호출하며 두번째 지령은 오직 Messenger창문만을 보여 준다. 어떤 경우에 꼭대기에 있는 차림표띠는 언제나 항목들중의 하나를 Communicator로 보여 준다. Navigator창문을 보는 경우에 Messenger를 능동으로 하려면 Communicator>Messenger를 리용한다(맨우에 있는 차림표띠에서 Communicator를 찰칵하고 그다음 차림표로부터 뒤따라 나오는 Messenger를 찰칵한다).

Messenger는 다목적우편의뢰기이다. 일반적인 우편처리기능은 제외하고도 우편을 내보내는 봉사기에 우편을 전송하도록 내부적으로 내장된 SMTP기능들을 리용한다. 또한 우편을 받아 들이는 봉사기로부터 우편을 가져 와서 그것을 우편함에 배포하는 POP의뢰기로서 동작한다. 한마디로 Messenger는 앞절에서 설명된 단일사용자에 기초한 일부 기능들을 처리한다. Messenger가 실행되고 있는 기계우에서는 우편을 보내고 접수하는것이 가능하다. 이 절은 초학도들과 Messenger를 지향하는 사용자들이 읽으면 재미 있을것이다.

13.7.1 Netscape Messenger의 구성

Messenger를 리용하기전에 그것을 설치해야 한다. 모든 Netscape설정은 Edit>Perferences로서 초기화된다. 어떤 새로운 창문이 화면우에 나타난다. 이때 왼쪽에 있는 삼각형기호를 찰칵함으로써 Mail &

Newsgroup종류를 확장한다. 이 차림표에서 항목들중의 어떤 주콤퓨터를 찾아야 하겠지만 이 절에서는 먼저 두가지 즉 Identity와 Mail Servers를 론의한다.

먼저 Identity에로 초점을 돌리자. 자기의 이름과 전자우편주소를 입력하여야 한다. Netscape가 .signature파일의 위치를 보여 준다는것을 이미 보았다. 이 파일은 Netscape로부터 보내지는 모든 우편통보문에 첨부할 내용이 들어 있는 파일이다.

다음 MailServers(그림 13-8)를 찰칵한다. 여기서 받아 들이고 내보내는 우편봉사기들의 주콤퓨터이름 혹은 FQDN를 입력해야 한다. 나가거나 들어 오는 우편이 또 다른 주콤퓨터 실례로 대학안의 주콤퓨터에 의해서 조종된다면 환경에 따르는 주소지정규칙에 따라 주콤퓨터이름 혹은 FQDN을 입력한다. 자기의 우편이 ISP에 의해서 조종된다면 ISP의 우편봉사기들의 이름을 입력한다. 그것은 자기 ISP의 우편봉사기가 인터네트상에 있으므로 FQDN으로 된다.

그림 13-8에 보여 주는 우편구성화면에서 우편을 받아 들이는 봉사기와 내보내는 봉사기의 령역이름들은 cal.vsnl.net.in으로서 꼭 같다. 그것들은 그렇게 될 필요는 없다. 자주 들어 오고 나가는 우편들은 다른 주콤퓨터들이 처리한다. 두 우편봉사기들의 사용자이름도 보여 준다. 개별적인 등록자리들에 대하여 이것은 일반적으로 전자우편주소의 부분적인 이름과 같다. 우편을 받아 들이는 봉사기에 대해서는 다음의것들중에서 한가지를 결심해야 한다.

- 통과암호는 매번 입력될 필요가 없도록 보관될수 있다.
- 우편은 규칙적으로 검사되며 새로운 통보문들이 자동적으로 내리적재된다.
- 통보문들은 봉사기에 남아 있어야 한다. 초학도들로서는 그것들이 있는가 확인해야 한다(기정적으로는 검사되지 않는다).

우편등록부 $HOME/nsmail이 여기에 설정되여 있는가를 이 화면으로부터 관찰하시오. 모든 우편처리기들이 같은 등록부를 리용하게 하고 싶다면 그것을 $HOME/Mail로 변화시킬수 있다. 이제는 Messenger를 리용하기 위한 준비가 되여 있다.

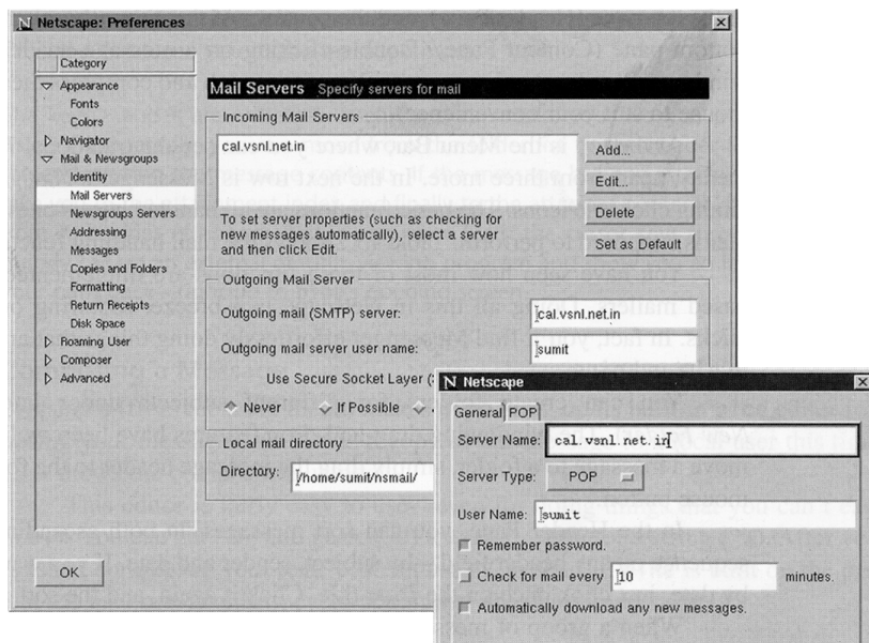



그림 13-8. Netscape우편봉사기의 구성화면

13.7.2 전경

Messenger의 특징은 아주 직관적이라는것이다. 그림 13-9에서는 4.5이상의 판본부터 3개 판으로 된 창문을 보여 준다. 모든 통보문등록부들은 읽은것과 읽지 않은 통보문들의 수와 함께 왼쪽에(통보문쎈터) 현시된다. 획이 굵은 등록부이름은 내부에 읽지 않은 통보문들이 있다는것을 의미한다. 여기서 통보문들은 두가지 부류 즉 email과 newsgroups로 나누어 진다.

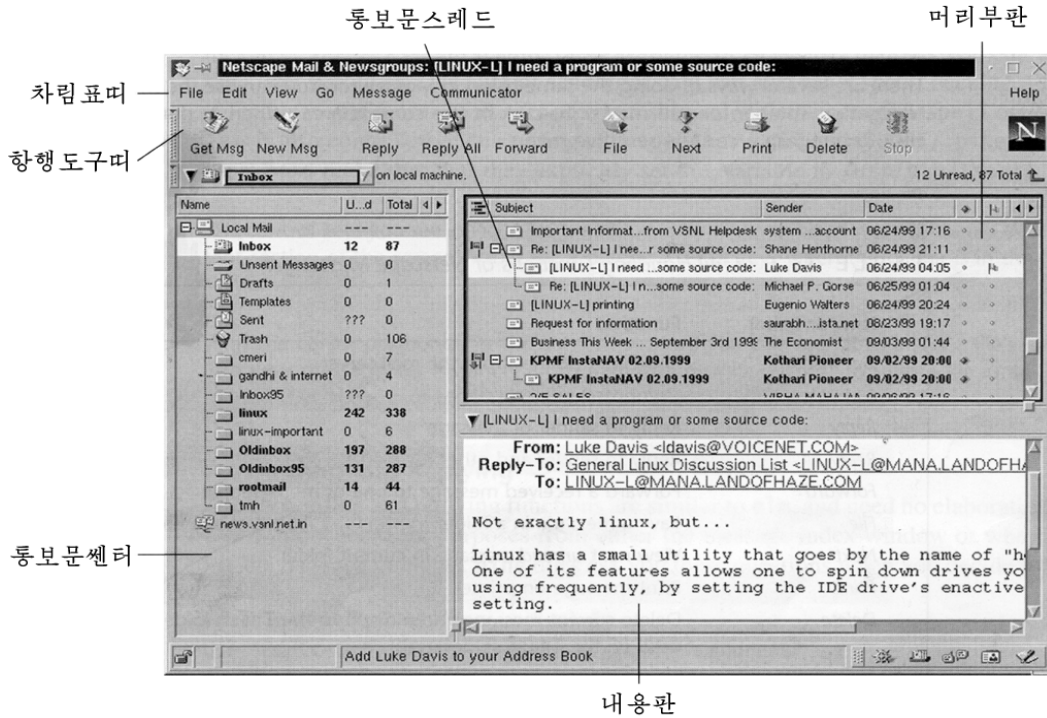



그림 13-9. Netscape Messenger

오른쪽의 통보문목록은 2개의 판으로 분할된다. 통보문머리부들은 우의 판(머리부판)에, 밑의 판에는 현재 선택된 통보문내용(내용판)들을 보여 준다. 통보문머리부에 두번 찰칵하면 더 큰 령역을 보여 주는 새로운 창문이 현시된다. 이 창문들은 마우스로 편리하게 넓힐수 있고 줄일수 있다.

우에 차림표띠가 있는데 일반적인 File, Edit, View차림표와 그외에 3개의 차림표들을 더 볼수 있다. 다음행은 찰칵가능한 아이콘들을 포함하고 있는 Messenger의 항행도구띠(Navigation Toolbar)이다. 기본적인 기능들은 모두 여기서 처리된다. 즉 어떤 우편처리기가 실행되도록 되여 있다. 표 13-2에 Messenger의 우편처리기능들을 보여 준다.

우리는 문자방식의 우편처리기들에서 대부분의 기능들이 수행되는 방법을 보았다. Netseape에서 하는 모든것들은 오직 간단한 마우스찰칵을 요구한다. 사용자는 실제로 elm과 pine에 전혀 불가능한것을 쉽게 하는 Messenger를 보게 될것이다. Local Mail(File>New Folder)밑에 서로 다른 주제를 위해서 등록부들을 만들수 있다.

통보문을 어떤 등록부에로 옮기려면 왼쪽마우스로 그 등록부에 통보문머리부를 간단히 끌기한다. 머리부판에서 통보문들을 어떤 머리부마당(Subject, Sender, Date)에 관하여 커지거나 작아 지는 순서로 분류할수 있다. 날자로 분류된 통보문들을 보고 싶다면 Date제목을 찰칵하시오. 그것을 다시 찰칵하면 분류순서는 반전된다.

표 13-2. Netscape Messenger의 우편처리기능들

아이콘 혹은 표식	기 능
Get Msg	우편봉사기로부터 통보문들을 검색한다
New Msg	우편통보문을 작성한다
Reply	통보문송신자에게 응답한다
Reply All	송신자와 다른 수신자들에게 응답한다
Forward	받은 통보문을 하나 혹은 그이상의 사람들에게 발송한다
File	Sent, Inbox, Unsent 등과 같은 폴더에 통보문을 보관한다
Next	현재등록부에서 읽지 않은 다음통보문을 보여 준다
Print	인쇄기로 통보문을 인쇄한다
Delete	현재통보문을 지우고 그것을 Trash폴더에로 옮긴다

어떤 통보문묶음이 또 다른 묶음의 호출에 따라 교체될 때 통보문스레드(thread)가 형성된다고 말한다. Messenger는 머리부판에 통보문들의 계층적인 구조로 나타나는 스레드들을 지원한다. 스레드들을 보려면 Sender제목의 왼쪽에 있는 아이콘을 찰칵한다. 두번 찰칵하면 배렬순서가 반전된다.

참고

Messenger에는 같은 기능을 수행하는 여러가지 방법이 있다. 차림표띠, 항행도구띠를 리용할 수 있고 혹은 문맥령역에서 오른쪽찰칵을 할수 있다. 의심스러우면 오른쪽단추를 찰칵해서 문맥의 존차림표를 보시오.

13.7.3 통보문의 구성과 접수

통보문을 구성하려면 New Msg를 리용하여 구성창문을 연다(그림 13-10).

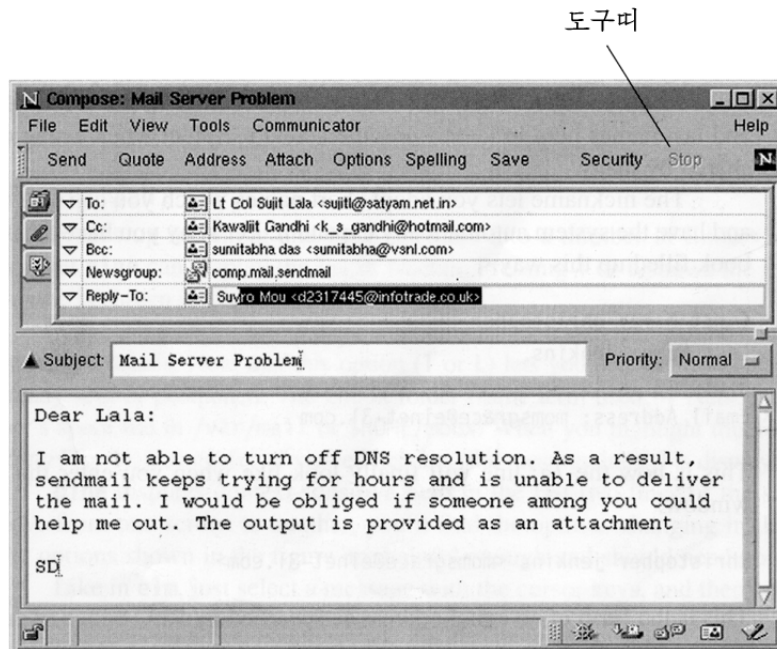



그림 13-10. Netscape Messenger의 구성창문

주소책(도구띠의 Adress)으로부터 주소를 꺼낼수 있으며 혹은 Messenger가 그것을 꼭같이 완성하도록 입력할수도 있다. 이것은 emacs사용자들을 격동시키는 아주 멋진 기능이다. 맞춤법을 검사할수 있

고 통보문과 함께 파일을 첨부물로 보낼수 있다.

Bcc:(Blind carbon copy)기능은 수신자들도 모르게 자기 통보문복사판에 자체로 표식을 붙여 준다. 통보문은 또한 새소식그룹에 주소화될수 있다(14.4). Messenger는 Reply-To:와 Followup-To:머리부들을 발생시켜 주는데 일부 다른 우편처리기들에서는 할수 없을것이다.

사용자는 통보문을 당장 보내거나(send) 혹은 연기(File>Send Later)할수 있다. 보낸 통보문과 보내지 않은 통보문은 개별적인 등록부에 따로 보관된다. 비직결상태에서 어떤 통보문묶음을 구성하고 어떤 묶음(File>Send Unsent Message)안에서 그것들을 후에 보낼수 있다.

통보문을 접수하려면 Get Msg를 찰칵하시오(그림 13-8). 통과암호를 보관하려고 선택하였다면 오직 한번만 통과암호를 물어 볼것이다. 모든 통보문들이 Inbox등록부안에 들어 오면 전송이 끝난다. 그다음 도구띠에서 관계되는 아이콘을 찰칵함으로써 응답하거나 통보문을 발송할수 있다.

13.7.4 주소책과 우편목록

주소책(Communicator>Address Book)은 Messenger에서 잘 수행된다(그림 13-11). 머리부판에서처럼 어떤 제목띠로 분류된 내용을 얻기 위하여 그 제목띠를 찰칵할수 있다. 주소들은 이름별명으로 혹은 전자우편주소를 기준으로 하여 커지거나 작아 지는 순서로 분류될수 있다.

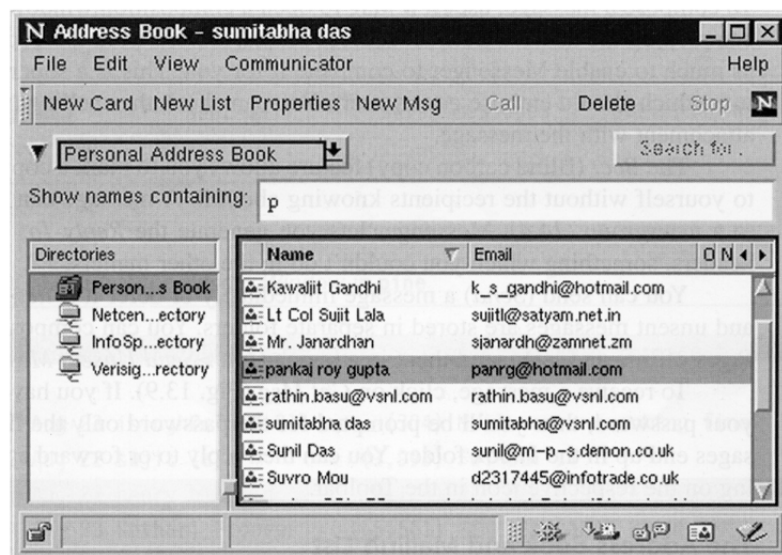



그림 13-11. Netscape Messenger의 주소책

누군가를 찾는다면 유일한 주소로 되는 2~3개의 문자만 입력한다. 여기서 우리는 어떤 항목을 지적하기 위하여 p를 입력한다. Messenger는 그 항목을 강조하고 여기서부터 우편통신을 시작하게 해준다. New Msg를 찰칵하면 구성창문이 나타난다.

주소책안에 어떤 내용을 만들 때 물론 성과 이름을 포함하는가를 확인하시오. 많은 사람들은 그것들을 제껴 놓고 간단히 전자우편주소를 입력한다. 일반적인 목적과 상반되게 이 이름들은 또한 통보문머리부안에 기입된다. 성과 이름은 독자적인 전자우편등록자리가 많이 공유될 때 필요한 수신자를 식별하는 것을 도와 준다.

별명은 To:혹은 Cc:안에 입력할수 있는 간단한 이름을 채용하게 하는데 자동적으로 체계가 그것을 확장한다. 다음과 같은 방법으로 채워 진 주소책안에 어떤 내용을 가지고 있다고 하자.

```
First Name: christopher
Last Name: jenkins
Nickname: kris
Email Address: momsgrace@elnet-31.com
```

다음의것은 그 창문에 kris문자렬을 입력할 때 To:행이 최종적으로 보여 준다.

```
christopher jenkins <momsgrace@elnet-31.com>
```

이것은 RFC822에 의하여 지적되는 전자우편주소형식이다. 수동적으로 주소책을 가득 채울 필요는 없다. 통보문을 접수할 때마다 통보문우에 오른쪽단추를 찰칵하고 Send to Address Book를 리용함으로써 주소책에 직접 송신자의(그리고 모든 수신자의) 주소를 보낼수 있다.

참고

여러개의 내용을 주소책에 가지고 있을 때 어떤 본문파일안에 (File>Export) 그것들을 보관할수 있다. 자기가 가는 곳마다에 이 파일을 가지고 다닐수 있으며 다른 기계우에서 이 목록을 얻을수 있다(File>Import).

주소책의 개념을 확장한것이 **우편목록**(mailing list)이다.

이 목록은 다중전자우편등록자리에 대한 별명으로 동작한다. 목록이름으로 주소화된 어떤 통보문은 자동적으로 그 모든 성원들에게 다 도착한다. 목록은 주소책에서 New List를 찰칵함으로써 호출된다. 목록에 이름, 별명을 주고 그다음 주소책으로부터 우편목록창문에로 끌기한다. 그러면 우편목록이 생성된다.

사용자가 어떤 통보문을 목록이름 혹은 별명으로 주소화하였을 때 Messenger는 자동적으로 그것을 실지주소로 바꾸어 대응시킨다. Netscape가 이 우편목록을 호출한다고 해도 실지로는 별명인데 사용자는 실지우편목록과, 자동적으로 우편을 돌려 보내는 기본기능 그리고 단순한 별명을 구분할수 있어야 한다.

13.7.5 스팸의 려과

자기가 적극적인 전자우편사용자이라면 우편함안에서 많은 스팸우편(spam mail)들을 보았을것이다. 대체로 사용자들은 일부 우편목록에 서명하거나 그것에 대해서 물을 때 모든 Web싸이트에 자기의 주소를 공개한다. 문자렬을 탐색하는 Messenger에 려과기능을 설정할수 있으며 그다음 지적한 방식대로 통보문에 작용한다.

탐색은 송신자의 전자우편주소, 통보문본체, Subject:행 혹은 Cc:행에서도 진행된다. 그 통보문들을 지울수 있고 다른 등록부로 옮길수 있으며 읽은 다음 그것들을 표식할수 있다. 스팸우편을 조종하기 위하여 지적할수 있는 많은 선택조건과 동작들이 있을수 있다.

Messenger는 오히려 스팸우편을 잘 처리한다. Edit>Message Filters를 선택하면 현재 존재하는 려과기들의 이름과 함께 창문이 나타난다(일부 경우에). 우리는 단어 Netphonic에 대한 주제 혹은 본체를 탐색하는 New(그림 13-2)로 새로운것을 만들수 있다. 우리는 이 통보문을 Trash등록부로 옮길수 있다. 가능한 탐색조건은 대부분의 과제들에 대해서 적합하며 탐색범위를 좁히기 위하여 5가지 조건을 지적할수 있다. 려과기를 능동으로 하기 위하여 려과목록의 3번째 렬우에 찰칵표식을 넣어야 한다(거기에 없는 경우에).

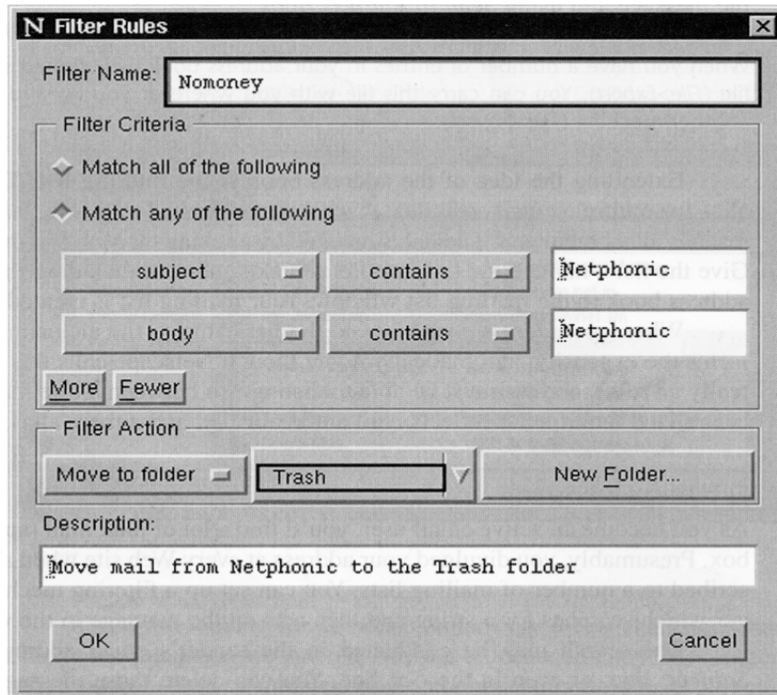



그림 13-12. Netscape Messenger: 통보문려과기의 생성

13.7.6 우편첨부물

SMTP통신규약은 매행에 1000개의 문자들로 제한되는 7bit본문파일만 처리한다. 그러므로 우편통보문과 첨부물(attachment)들은 전통적으로 본문제한을 받는다. 즉 2진파일은 포함할수 없다. 오늘날에는 이러한 상황이 변화되였다. Messenger는 elm과 pine이 가지고 있는 결함인 본문첨부물들은 물론 2진첨부물들도 다 처리한다.

pine과 elm도 본문첨부물들을 보기 할수 있다. 그러나 Messenger는 그것들을(기본통보문도 포함해서) 직접삽입(inline)으로 보기 할수 있다. Messenger도 한 걸음 더 전진해서 GIF와 JPEG형식으로 된 도형들을 직접삽입으로 현시한다.이 두가지 형식은 Web상에서 표준이며 모든 열람기는 도움말을 보지 않고도 그것들을 현시할수 있다. pine은 그림을 보기 위하여 외부프로그람을 호출해야 한다.

파일을 첨부물로 보내려면 구성창문(composition window)으로부터 Attach를 선택한다. 그다음 파일을 고르고 open을 찰칵한다(아마 이렇게 하면 Attach가 호출되였을것이다). 통보문이 보내질 때 다중부분통보문(multipart message) 즉 기본통보문과 첨부물을 포함하고 있는 단일한 통보문으로서 보내진다. 이 장의 마지막절에서 이 통보문의 구조를 결정하는 명세서의 일부 중요한것들을 론의할것이다.

접수하는쪽에서 이 우편을 내리적재 할 때 머리부가까이에 아이콘이 나타난다. 그것이 Messenger가 처리할수 있는 첨부물이라면 파일의 내용은 그사이에 분리선과 기본우편통보문밑에 나타난다. 첨부물이 개별적인 파일로 보내졌다 할지라도 실지로는 통보문본문과 첨부되는 파일의 내용들을 포함하고 있는 단일한 통보문으로 보내진다는것을 알아야 한다. 이 첨부되는 파일은 그림일수도 있고 음성, 비데오파일일수도 있다.

Messenger가 첨부물을 보기할수 없다면 방조응용프로그람(helper Application)을 호출한다. 이 응용프로그람은 Messenger와 관계가 없는 순전히 외부응용프로그람이다. 쉘프롬프트로부터 독립적으로 이 응용프로그람을 호출할수도 있다. 방조응용프로그람을 식별하고 호출하는 기술은 마지막절에서 설명된다.

13.8 콤퓨터앞을 떠날 때(vacation)

자기의 우편함안에 우편이 쌓이지 않도록 하기 위하여 다른 사람들에게 자기가 장기휴가를 받으려 한다는것을 알려 주어야 할 때가 있을수 있다. 버클리의 vacation지령은 .forward파일의 우편발송기능을 리용한다. 이 지령은 오직 수신자가 휴가를 받았을 때(혹은 다른 목적으로)에만 리용된다.

SVR4 vacation지령은 대체로 버클리의 판본과 서로 다른 문법을 리용하고 있으므로 미움을 받고 있다. Solaris와 Linux는 인수없이 호출되는 BSD판본을 리용한다.

```
vacation
```

이것은 다음의 통보문 혹은 그와 류사한것을 포함하는 편집기를 불러 들인다.

```
Subject: away from my mail
I wil not he reading my mail for a while. Your
mail regarding "$SUBJECT" will be
read when I return.
```

이 통보문을 편집할수 있지만 자동적으로 vacation에 의해 발생되므로 Form:과 To:행을 반드시 추가할 필요는 없다. 편집기를 완료할 때 이 4개 행은 $HOME/.vacation.msg.안에 보관된다. vacation은 .forward파일안에 다음의 등록항목을 넣는다.

```
\sumit, "|/usr/bin/vacation sumit"
```

여기에는 우리가 .forward와 함께 리용하고 있는 여러가지 형식이 있다. Sumit로 주소화된 통보문들은 .vacation.msg안에 보관된 통보문을 모든 송신자들에게 보내주는 vacation프로그람에로 발송된다. 그러나 들어 오는 모든 통보문들은 보관될것이다. 즉 여기서 /sumit는 통보문의 복사판이 /var/mail/sumit안에 보관된다는것을 지적한다.

여기저기에 .vacation.msg와 .forward를 가지고 있을 때 -I선택항목과 함께 vacation을 실행시켜야 한다. vacation을 해제하려면 .forward파일을 제거하거나 이름을 고쳐야 한다.

Solariss체계에서 vacation은 두개의 추가적인 파일 즉 vacation.pag와 .vacation .dir파일들을 만든다. 이 파일들은 vacation이 가능할 때 통보문들을 보낸 송신자들의 목록을 가지고 있다. vacation -I는 이 파일들을 지운다.

13.9 2진파일의 처리(MIME)

우편읽기응용프로그람이 특별한 파일을 처리할수 없다면 어떤 일이 일어 나겠는가? 그리고 ASCII코드의 8번째 문자를 리용할수 없을 때 SMTP통신규약은 다매체첨부물들을 어떻게 처리하는가? 오늘날에는 이 첨부물들을 처리하는 특별한 통신규약인 **다목적인터네트우편확장**(Multipurpose Internet Mail Extensions:MIME)이 있는데 지금 인터네트상에서 표준적으로 리용되고 있다.

MIME는 2진파일과 여러가지 자료형식을 단일한 통보문으로 포함하기 위하여 우편에 대한 정의를 확장한다. 이전의 통신규약 RFC822는 통보문머리부와 관계되지만 MIME는 내용의 형식에 관계된다. MIME는 행길이에 제한이 없다(RFC822는 1000개의 문자로 제한된다). MIME는 통보문을 보내기전에 부호화하는데 이 자료를 그것을 처리하는 방법에 관한 정보와 함께 보낸다.

이 자료는 그후에 MIME용으로 우편을 읽는 사람에 의해서 그쪽에서 복호된다. 량쪽의 사용자대행체들이 MIME를 리해한다면 MIME통보문들은 쉽게 그것들사이에 교환될수 있다. 접수하는 측의 우편읽기프로그람은 그것을 현시하기 위한 능력을 가질수도 있고 혹은 그 일감을 수행할수 있는 방조프로그람(외부프로그람)을 참조할수 있다.

두가지 중요한 MIME머리부들

첨부물을 보낼 때 다중부분통보문은 명백한 구분을 보여 준다. 첫번째 부분은 뒤에 통보문본문이 따르는 2개의 머리부를 보여 준다. 두번째 부분(여기서는 첨부되는 JPEG파일 즉 화상을 서술하는)도 역시 자기의 머리부들을 가지고 있다. 이 머리부뒤에 부호화된 자료들이 뒤따른다. 첨부물을 포함하고 있는 우편통보문의 어느 한 부분을 그림 13-13에서 보여 준다.

```
This is a multi-part message in MIME format.
---------------4369A7A6B4A758020BD17F87
Content-Type: text/plain；charset=us-ascii
Content-Transfer-Encoding： 7bit

Here you can see the picture of marc andreesen. Did you know that he
designed the browser at Illinois University for $6.85 an hour?
---------------4369A7A6B4A758020BD17F87

Content-Type: image/jpeg;
Name="andreesen.jpg"
Content-Transfer-Encoding: base64
Content-Disposition:inline;
Filename="andreesen.jpg"

/9j/4AAQSkZJRgABAgEASABOAAd/7RfQUGhvdG9zaG9wIDMuMAA4QklNBAQAAAAAAwOcAgAA
AgAVHAJ4AkBNYXJjIEFuZHJlZXNlbiwgTmVOc2NhcGUgQ29tbXVuaWNhdGlvbnMgY28tZm91
BmRlciBhbmQgc2VuaW9yIHZpY2UgcHjlc2lkZW50IG9mIHRlY2hub2xvZ3ksIHBhdXNlcyBk
DCBicm93c2VyLCBhbm5vdW5jZWQgYSBtYWpvciBleHBhbnNpb24gb2YgaXRzIGxpbmUgb2Yg
... 64진법으로 부호화된 자료 ...
```

그림 13-13. MIME형식으로 첨부물을 포함하고 있는 우편통보문

여기에는 모든 우편통보문들에서 일반적으로 보이는 2개의 MIME마당 즉 Content-Type:과 Content-Transfer-Encoding:이 있다. 그것들은 서로 다른 부분에 따라 내용도 서로 다르다는것을 주의하시오.

내용형(Content-Type)머리부는 개별적인 부분의 통보문 본체안에 있는 자료의 형을 정의한다. 이 형은 두가지부분 즉 기본형과 보조형으로 구성된다. 기본형은 본문, 화상, 음성, 비데오일수 있다. 기본형은 /이 뒤따르며 그다음 보조형이 뒤따른다. plain은 본문의 보조형이다. 일반적으로 하나의 형은 다중보조형을 가진다. 실례로 화상(image)형은 보조형으로서 jpeg 혹은 gif를 가질수 있다. 비데오자료는 audio/aiff가 음성파일로 리용되는 동안 video/mpeg로 표현될수 있다.

MIME는 또한 **내용전송부호화**(Content-Transfer-Encoding) 즉 자료를 부호화하는데 리용되는 부호화기술을 정의하는 또 다른 머리부를 지적한다. 일반적으로 리용되며 잘 알려 진 3가지 부호화기술이

있다. ASCII자료에 대해서는 7bit와 인용부호화된 인쇄가능형(quoted-printable)을 리용하며 64진수는 화상들, 음성과 비데오자료를 부호화한다. 64진수는 3개의 8bit자료(octet)를 4개의 6bit문자들로 부호화한다. 따라서 파일의 크기가 33%로 증가한다. 통보문의 두번째 부분의 내용은 andreesen의 화상을 포함하고 있는 자료를 64진수로 부호화한것을 보여 준다.

이 모든것은 무엇을 의미하는가? Messenger는 JPEG2진파일을 직접삽입(inline)으로 즉 방조응용프로그람이 없이 보기한다. 그러나 사용자대행체들인 pine과 elm은 조수체계(assistance)가 없으면 이 파일들을 보기할수 없다. 이 조수체계는 metamail이라고 부르는 다매체우편처리기에 의해 부분적으로 제공된다. 이 프로그람은 특별한 내용형머리부를 처리하는데 리용되여야 하는 방조응용프로그람을 결정하기 위하여 /etc/mailcap파일을 읽는다.

pine은 metamail기능을 내장하고 있다. 그러나 이것도 etc/mailcap파일을 읽는다. 어떤 체계상에서 내용형머리부로 image/jpeg를 가지고 있는 자료를 조종하기 위하여 어느 응용프로그람을 호출하겠는가를 결정하는데 관계되는 행은 다음과 같이 보여 준다.

```
image/jpeg; xv %s; test=test -n "$DISPLAY"
```

이것은 pine이 내용형머리부를 보기 위하여 xv화상프로그람을 리용하겠다는것을 의미한다. %s는 파일이름을 위한 장소유지자(placeholder)로 동작한다. 통보문을 보고 있을 때 그림 13-14에서 보여 주는 것처럼 부분목록을 보려면 v를 눌러야 한다.

```
PINE 4.10    ATTACHMENT INDEX       Folder：INBOX Message 67 of 68

 1       1 lines     Text/PLAIN
 2     17 KB         Image/JPEG
```

그림 13-14. pine으로 현시한 첨부물색인

화상부분을 선택하기 위하여 강조색을 움직이는 경우 다시 [Enter]를 누르거나 v를 누른다. 그러면 xv방조프로그람 즉 어떤 특별한 창문에 andreesen.jpg화상파일을 현시하는 프로그람이 기동한다. 그러나 pine은 처음에 자료를 부호화하는데 리용된 기술(64진법)로 내용을 복호하여야 한다(X에서는 화상을 보려면 xterm창문으로부터 pine을 리용해야 한다).

Netscape는 MIME를 내장하고 있다. 또한 자기의 MIME파일묶음을 $HOME/.netscape등록부안에 가지고 있다. MIME는 또한 Web에서 중요한 역할을 하고 있는데 다음장에서 론의할것이다.

요 약

전자우편은 값이 눅고 비공식적이며 접속되지 않은 상태에서도 빠른 통신방법이다. 어떤 전자우편주소는 사용자가 같은 주콤퓨터상에 있다면 오직 사용자이름으로 구성될수 있고 사용자이름@주콤퓨터이름 혹은 사용자이름@령역이름형식을 가질수 있다. 두번째것은 인터네트상에서 리용된다.

들어 오는 우편은 /var/mail/(SVR4) 혹은 /var /spool/mail(Linux)안에 사용자ID로 이름 지어 진 본문파일(우편함)에 추가된다. 초기의 우편처리프로그람은 mail이지만 elm과 pine은 더 위력하며 사용자들에게 친근하다. 한번 본 우편은 대체로 $HOME/mbox안에 보관되지만 elm과 pine은 $HOME/Mail안

에 있는 개별적인 폴더를 리용한다.

우편으로 응답할수 있고 간단한 건누르기로 우편통보문을 발송하고 지우며 보관할수 있다. 어떤 패턴을 탐색할수 있으며 또한 스팸을 처리한다. pine을 리용하여 우편보내기를 연기할수 있으며 주소책을 가지고 있을수 있고 통보문들에 대한 맞춤법검사를 할수 있다. pine은 기동한 상태에서 자동적으로 건누르기를 실행하여 구성될수 있다.

elm은 $HOME/.elm/elmrc파일을 편집함으로써 구성될수 있다. pine은 $HOME/.pinerc를 리용한다.

$HOME/.forward안에 필요한 전자우편주소를 넣음으로써 자기우편의 방향을 돌릴수 있다. .signature파일에 개인세부자료를 넣음으로써 모든 통보문에 그것들이 붙는가를 확인할수 있는데 모든 우편처리기들은 기정적으로 리용한다.

우편은 단순우편전송규약(Simple Mail Transfer Protocol:SMTP)을 리용하여 전송된다. SMTP는 넘겨 주어야 할 대행체에 통보문을 넘겨 준다. 우편은 원격싸이트로부터 우편국규약(Post Office Protocol:POP)(전화회선에서 자주 리용된다.)을 리용하여 가져 오게 된다.

Netscape Messenger는 우편을 보내고 받는 자기의 SMTP와 POP의뢰기기능을 리용한다. 우편통보문들은 어떤 순서로 보관될수 있으며 혹은 마우스를 리용해서 그 등록부를 옮길수 있다. Messenger는 또 다른 곳으로부터 어떤 스레드로 보내는 통보문들을 묶을수 있다. 그리고 GIF와 JPEG도형들을 직접 삽입으로 보기(통보문과 함께)할수 있다.

vacation지령은 모든 사람들(휴가상태에 있는 사용자에게 어떤 통보문을 보내는)에게 이미전에 만든 통보문을 보낸다. 그 지령은 사용자의 .forward파일로부터 실행된다.

인터네트우편은 다매체첨부물을 전송할수 있게 하기 위하여 다목적인터네트우편확장(MIME)을 리용한다. MIME는 내용을 처리해야 할 방조응용프로그람을 결정하기 위하여 내용형머리부를 리용한다. 내용전송부호화는 리용된 부호화기술을 정의한다. 64진법은 화상, 음성, 비데오파일을 부호화하는데 리용되는 부호화기술이다.

시험문제

1. 전자우편에서 리용되는 용어로서 폴더란 무엇인가?
2. $HOME/mbox안에 어떤 통보문이 있는것을 보았다. 그것은 무엇을 의미하는가?
3. 우편을 읽는 사람들이 어디로부터 모든 우편을 얻는가?
4. 사용자이름과 함께 mail을 호출하는것과 사용자이름없이 실행시키는것과의 차이는 무엇인가?
5. mail을 리용해서 어떤 사용자에게 모든 체계처리들을 보여 주는 ps출력을 보내시오.
6. elm의 구성파일은 어디에 보관되여 있는가?
7. pine으로 첨부물을 어떻게 보내는가?
8. 왜 elm과 pine은 인터네트상에서 쓸모 있는가?
9. vacation기능을 어떻게 해제하는가?
10. MTA는 누구에게 우편을 넘겨 주는가?
11. Messenger만을 꺼내려면 Netscape를 어떻게 호출해야 하는가?

련습문제

1. elm이나 pine들과 같은 우편처리기들은 우편함의 위치를 어떻게 알고 있는가?
2. 사용자 john에게로 주소화된 우편통보문이 To:행에 bill johnston<john@planets.com>를 보여 준다면 그 통보문은 john에게 도착할것인가?
3. 같은 주콤퓨터상의 100명의 사람에게 통보문복사판을 보내려고 하는데 어느것이 자기의것과 차이나며 복사판은 어디서 만들어 지는가?
4. elm에서 머리부에 prize문자렬을 포함하고 있는 모든 통보문들을 우편함안에서 지우려면 어떻게 해야 하는가?
5. 어떤 우편처리기가 통보문의 형식을 리해할수 없다면 무엇을 해야 하는가?
6. Netscape로 보내진 모든 통보문들을 elm을 리용하여 어떻게 읽을수 있는가?
7. elm에서 통보문을 작성하기 위하여 vi편집기를 리용하고 싶다면 무엇을 설정해야 하는가?
8. elm을 리용하여 어떤 우편이 있는가 없는가를 어떻게 검사하는가?
9. elm에는 없는 pine의 3가지 기능들을 지적하시오
10. 주차림표가 아니라 주소책을 먼저 보여 주도록 pine을 어떻게 호출할것인가?
11. 보내지는 통보문에 주소책과 전화번호를 같이 보내려 한다면 어떻게 해야 하는가?
12. 원격주콤퓨터에 있는 자기의 등록자리로 우편을 보냈는데 우편이 자기에게 도착하지 못했다. 대신에 자기가 등록자리를 가지고 있는 또 다른 주콤퓨터에서 전송은 끝났다. 무엇이 원인으로 될수 있는가?
13. POP3과 SMTP는 어떻게 차이나는가?
14. 보통환경에서 목적지에 통보문이 도착하는데 시간이 얼마나 걸리는가. 왜 그런가?
15. 전화회선으로 련결된 원격주콤퓨터로부터 우편을 내리적재하는데 리용되는 두가지 통신규약은 무엇인가?
16. 전자우편과 새소식그룹의 문맥에서 나오는 스레드란 무엇인가?
17. 다른 수신자는 모르게 통보문의 복사판을 누군가에게 보내려고 한다. 그렇게 할수 있는가?
18. 전자우편에서 리용되는 용어로서 우편목록이란 무엇인가?
19. 통보문에 한개의 그림파일과 한개의 음성파일을 첨부물로 보낼 때 그것들은 자기의 기계들을 어떻게 떠나는가?
20. 휴가상태가 아닐 때 vacation은 어느 두 파일을 변경시키는가?
21. 방조응용프로그람이란 무엇인가? 그리고 우편을 조종하는데서 방조응용프로그람의 역할은 무엇인가?
22. 다매체첨부물들을 우편으로 보내는데 리용되는 통신규약은 무엇인가. 왜 특별한 통신규약이 필요되는가?
23. MIME는 왜 내용형의 자료를 다른쪽으로 보내는가?
24. 무슨 기술이 2진파일을 부호화하는데 리용되며 파일의 크기에 어떻게 작용하는가?
25. pine으로 어떤 GIF첨부물을 보려고 하는 동안에 다음의 통보문을 얻었다.

    ```
    Don't know how to display Application/OCTET-STREAM attachments. Try save..
    ```

    pine은 무슨 말을 하려고 하며 이때 무엇을 해야 하는가?
26. 어느 파일이 우편처리기가 조종할수 없는 파일을 보아야 할 때 방조응용프로그람을 결정하도록 보여주는가?

제 14 장. 인터네트

인터네트는 ARPA(Advanced Research Projects Agency)가 콤퓨터에 기초한 믿음직한 통신체계를 개발하는 과정에 출현하였다. ARPA는 미국방성의 연구개발팀이다. 인터네트는 TCP/IP망들의 집합인데 흔히《망들의 망》이라고 한다. 인터네트는 또한 TCP/IP의 파케트절환기술을 리용하여 개방형구조의 망작업을 구현하기 위한 콤퓨터력사에서 의뢰기-봉사기모형의 가장 큰 실험이다.

UNIX의 소문이 자자한것의 주되는 요인의 하나는 그것이 세계적으로 인터네트를 추동시킨 통신기술의 전진을 위하여 20여년동안 지배적인 역할을 하였기때문이다. UNIX는 TCP/IP도구들의 발전을 위한 가동환경으로 리용되였으며 인터네트상에서 리용되는 모든것을 봉사한다. 오늘 대부분의 인터네트봉사기들은 UNIX와 Linux기계상에서 그 봉사들을 제공한다. UNIX는 인터네트의 언어이다.

인터네트상에서 리용되는 일부 TCP/IP응용프로그람들은 제11장에서 이미 론의되였다. 이 장에서는 World Wide Web를 비롯하여 인터네트에 대해서 론의한다. 여기서는 문자방식과 도형방식으로 혼합된 도구들을 고찰하며 Web를 다루는데서 Netsape를 제한적으로 리용한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 인터네트령역들의 기관과 그 이름짓기방법을 배운다(14.1).
- 우편물목록을 리용하는 방법을 알아 본다(14.3).
- 망새소식이 어떻게 작업하며 새소식그룹이 어떻게 조작되는가를 리해한다(14.4).
- 새소식그룹을 처리하는 tin과 Netscape Messenger를 리용한다(14.5, 14.6).
- irc지령을 리용하여 여러명의 사람들과 직결IRC대화를 진행한다(14.7).
- World Wide Web가 떨어 져 있는 기계들에 문서들을 련결해 주는 방법을 배운다(14.8).
- Netscape Navigator열람기의 여러가지 구성요소들을 해석한다(14.9).
- Web가 하이퍼본문전송통신규약(HTTP)을 리용해서 어떻게 작업하는가를 파악한다(14.10).
- Web가 도형과 본문을 함께 가지는 하이퍼본문표식달기언어(HTML)를 어떻게 리용하는가를 리해한다(14.11).
- Web페지와 그의 도형들을 보관한다(14.12).
- 열람기의 성능을 높인다(14.13).
- MIME기술이 Web에 어떻게 리용되는가를 배운다(14.15).

참고

설계자들이 제작한 인터네트의 세부적인 력사는 Web싸이트주소 http:w w w. isoc.org/internet/history/brief.html.에서 찾아 보시오. history등록부에 또 다른 흥미 있는 문서들이 있다.

14.1 인터네트의 웃준위령역

인터네트는 TCP/IP망에서 리용되는것과 같은 간단한 단어로 된 주콤퓨터이름들을 리용하지 못한다. 주콤퓨터들은 령역들에 속하며 이 령역들과 부분령역들은 인터네트의 골격을 구성한다. 인터네트는 초기에 3개의 문자로 된 8개의(지금은 그이상) 웃준위령역들을 가지고 있다. 오늘 모든 나라들은 이 준위에서 한개의 령역을 가지고 있다(표 14-1). 웃준위령역들은 거꾸로 된 나무처럼 편성되였는데(그림 14-1) 모든

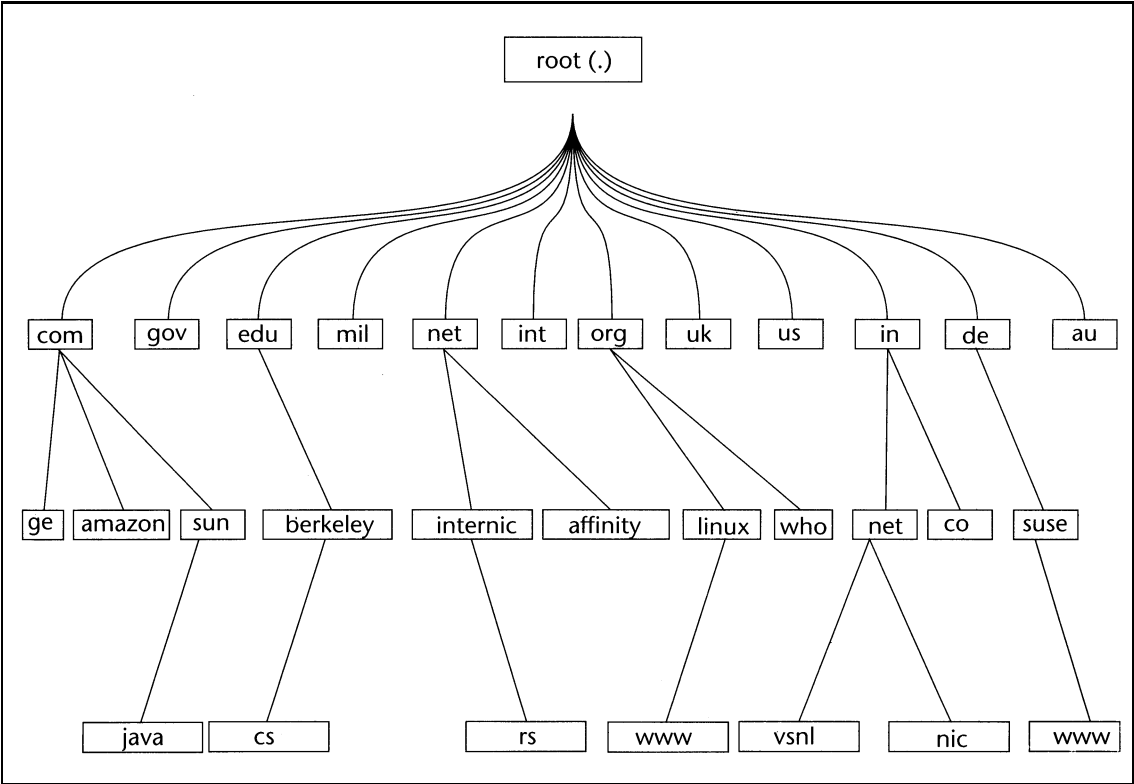
령역들은 자기의 부분령역들을 가지고 있다.

표 14-1. 인터네트의 웃준위령역들

령역이름	표 현
int	국제적인 기관들
edu	교육기관들
gor	미국정부기관들
mil	미국군부기관들
org	단체들
com	상업적인 기관들
net	망관련기관들
arpa	역분석을 위한 령역
uk,ch,us,de등	영국, 스위스, 미국, 도이췰란드 등

이 3개의 문자로 된 령역들을 **일반령역** 혹은 **기관령역**(orgarizational domain)이라고 한다. MIT는 edu령역에, 싼 마이크로시스템즈는 com령역에 그리고 InterNIC는 net령역에 속한다. 인터네트가 미국에서부터 기원되였으므로 미국의 기관들은 이처럼 일반령역(generic domain)에 위치를 정해야 할것 같지만 사실 mil과 arpa를 제외하고 다른 미국의 기관들은 이러한 일반령역에 위치를 두고 있지 못하고 있다.

다른 나라의 기관들도 이러한 령역들에 속해 있다. 모든 나라들은(미국도 포함) 2개의 문자로 된 령역이름으로 표현된다. 영국(gb대신에 uk를 리용)은 별개로 하고 모든 이름들은 ISO-3166규격으로부터 얻어 진다. 도이췰란드는 de를, 프랑스는 fr, 인디아는 in, 일본은 jp를 가진다. 이 나라들의 령역들은 일반령역들로서 같은 계층구조를 가지고 있다.

그림 14-1. 인터네트령역나무

부분령역협약은 나라마다 다르다. 일부 나라들은 일반령역들과 류사한 구조를 리용하여 지역에 따라, 일부는 기능에 따라 그것들을 편성한다. 독자들은 많은 나라들의 웃준위령역밑에 co와 com령역 (mirc.co.uk, sun.com.sg와 같이)들을 볼수 있다. us령역은 미국의 매개 주에 해당한 50개의 부분령역들로 령역을 리용한다.

14.2 인터네트봉사

인터네트봉사는 날을 따라 계속 발전되여 왔다. 그의 일부는 BSD UNIX로 통합되기전에 DARPA의 후원하에 BBN(즉 Bolt, Beranek, Newman)에 의하여 개발되였다. 아래에 제시된 응용프로그람들중 일부는 UNIX의 표준기능으로서가 아니라 Web를 목적으로 독립적으로 분리되여 개발되였다. 현재 이것들은 인터네트에서 리용되고 있는 응용분야들이다.

- 전자우편: 개인용통신을 목적으로 가장 광범히 리용된다.
- 텔네트(telnet): 원격콤퓨터에 접속하여 그것을 국부콤퓨터로서 취급할수 있게 한다.
- FTP: 두대의 콤퓨터들사이에 파일을 전송할수 있게 한다.
- 우편목록: 어떤 사람들의 그룹에 자동적으로 통보문을 낸다.
- 새소식그룹: 사용자들에게 특정한 주제의 정보들을 보내고 공유할수 있게 한다.
- 인터네트중계담화(IRC): talk와 류사한 직렬담화를 보장한다.
- World Wide Web: 문서와 화상들이 서로 련결된 하나의 방대한 정보보관고로서 여기서는 한 문서가 다른 문서로부터 접근될수 있다.

전자우편과 ftp, telnet에 대한 명세는 톰슨과 리치에가 UNIX의 첫 판본을 보여 주기 바로 한해전인 1972년에 제출되였다. 이것들은 독립적인 망들에서 유용하므로 그 특징들에 대해서는 이전 장에서 이미 론하였다. 새소식그룹, IRC, Web는 인터네트계에 새롭게 추가되였다. 표 14-2는 현재 사용되고 있는 인터네트봉사들과 그에 따르는 의뢰기들을 보여 준다. 현재 이 표에서는 도구 archie와 gopher는 제시하지 않는다.

표 14-2. 인터네트응용프로그람들

응용프로그람/규약	문자에 기초한 의뢰기	GUI의뢰기
파일전송규약(FTP)	ftp지령	(1) Netscape Navigator에서의 URL앞붙이 ftp:// (2) xftp
텔네트규약	telnet지령	(1) Netscape Messenger (2) Netscape Navigator에서의 앞붙이 telnet://
전자우편/SMTP	mail, pine, elm지령	(1)Netscape Messenger (2)Net scape Navigator에서 통보문을 보내기 위한 앞붙이 mailto:
우편목록(mailling list)	임의의 전자우편의뢰기	임의의 전자우편의뢰프로그람
새소식그룹/NNTP	tin과 trn지령	(1) Netscape Messenger (2)Netscape Navigator에서 newgroup에 가입하기 위한 URL앞붙이 news:
인터네트중계담화(IRC)	irc지령	xirc
WWW(HTTP)	lynx지령	Netscape Navigator(netscape지령)

초기에 이러한 응용프로그람들은 문자에 기초한 도구들을 사용하였으며 매 도구들은 자기자체의 내

부지령들을 가지고 있다. 이 도구들은 보통 기능적으로 제공되지만 후에 출현한 도형처리쏘프트웨어는 사용하기가 대단히 편리하다. 이 장에서는 이러한 프로그람들의 일부분으로서 UNIX체계에서 가장 광범히 리용되는 도형인터네트도구인 Netscape Communicator에 대하여 론의한다.

14.3 우편목록

우편별명(mail alias)은 여러 사람들에게 어떤 통보문을 보내도록 해준다. 추가와 삭제에 의한 변경들은 별명이 리용되는 모든 장소에 유지되여야 한다. 한편 우편목록은 집중적이며 통보문들은 자동적으로 보내진다. 사람들이 흥미를 가지는 모든 주제들을 포함하는 인터네트에서 이러한 목록들은 100,000여개가 넘는다.

사용자는 자기가 흥미를 가지는 주제에 해당한 우편목록에 가입(지불액을 요구하지 않는 형태)할수 있다. 목록의 한 성원에 의하여 보내진 질문은 그 목록의 모든 성원들에게 다 전달되며 우편함에는 규칙적으로 통보문들이 쌓이게 된다. 사용자는 자기가 가입한 목록이 불필요하다고 생각되면 목록에서 탈퇴(unsubscribe)할수도 있다. 목록은 조절(moderate:사람에 의한 관리)될수 있고 자동화(프로그람에 의한 처리)될수 있다.

14.3.1 가입과 탈퇴

lynx라는 본문열람기에 대한 man페지들을 본다면 우편목록의 가장 일반적인 절차에 대하여 서술한 아래와 같은 세개의 단락을 보게 될것이다.

```
If you wish to contribute to the further development of Lynx, subscribe to our m
ailing list. Send email to <majordomo@sig.net> with "subscribe lynx-dev" as the
only line in the body of your message.

Send bug reports, comments, suggestions to <lynx-dev@sig.net> after subscribing.

Unsubscribe by sending email to <majordomo@sig.net> with "unsubscribe lynx-dev"
as the only line in the body of your message. Do not send the unsubscribe messag
e to the lynx-dev list,itself.
```

우에서 보는바와 같이 모든 우편목록은 주제와 관련되는 자기의 이름을 가진다. 여기서 lynx-dev는 Lynx쏘프트웨어를 론의하는 목록이다. 그것은 majordo ms라는 프로그람에 의하여 관리된다. 사용자는 목록에 질문들을 보내기전에 먼저 majordomo@sig.net(목록의 관리주소)에 아래와 같은 간단한 통보문을 보내는것으로써 목록에 가입해야 한다.

```
subscribe lynx-dev              관리자에게 보내기 위하여
```

Subject:행을 공백으로 하시오. 대부분의 우편목록프로그람들은 거기에 무엇을 놓든지간에 관계하지 않는다. Majordomo프로그람은 지령과 비슷하게 사용자가 본체에 서술한 모든 행의 통보문들을 분석한다. 이 프로그람은 목록에 사용자의 우편주소를 추가하고 자동적으로 확인통보문을 보낸다.

목록에 가입한 다음 사용자는 질문을 보낼수 있으며 이때 질문들은 관리주소가 아니라 목록주소로서 주소화되여야 한다. 즉 여기서는 그러한 주소로서 lynx-dev@sig.net이다. 목록봉사기는 통보문의 복사판

을 만들어 그것을 목록의 모든 성원들에게 보낸다. 즉 봉사기는 통보문들을 보내기 위한 하나의 반사기로서 동작한다.

목록에서 탈퇴하기 위하여서는 아래와 같은 통보문을 관리자에게 보내여야 한다.

unsubscribe lynx-dev　　　　　관리자에게

주해

목록에 어떤 주제와 관련되는 자기의 통보문을 보내기 위하여서는 목록주소를 리용하여야 하며 관리주소는 목록에 가입하거나 탈퇴하는 경우에 리용한다. 목록주소는 일반적으로 전자우편주소(여기서는 lynx-dev) 의 사용자요소로서 특정한 주제를 특징 짓는다.

참고

일반적인 질문들에 대한 대답은 대체로 새소식그룹에 미리 정해 져 있다. 그러므로 이 경우에는 그와 관계되는 FAQ(14.4)를 탐색하는것이 더 좋다.

14.3.2 listserv프로그람들

현재 리용되고 있는 대표적인 우편목록프로그람들에는 listserv, listproc , majordo mo가 있다. 우와 같은 프로그람들을 listserv프로그람들이라고 부르는것은 listserv프로그람이 제일 초기에 나왔기때문이다. 매개 프로그람들은 자료를 보내는것은 물론 관리할수 있는 능력도 가지고 있다.

listserv프로그람들은 많은 내부지령들을 가지고 있다. 대표적으로 subscribe와 unsubscribe를 들수 있다. 우의 봉사프로그람들이 리용하는 지령들은 일반적으로 같지만 그 일부는 다른 목적으로 리용된다. 실례로 지령 unsubscribe는 listproc와 majordomo에서만 사용하며 listserv에서는 그와 같은 기능을 수행하는 signoff지령을 리용한다.

만일 우편목록에 대하여 처음이고 listserv@lists.colorado.edu와 같은 어떤 봉사기프로그람에 대한 주소를 가지고 있다면 이러한 주소에 아래와 같은 두 행의 통보문을 보내야 한다.

help　　　　　봉사기가 인식하는 지령들에 대한 목록
lists　　　　　봉사기에 의하여 관리되는 모든 목록

자기의 통보문에 우와 같은 지령들을 놓는다면 사용자는 두개의 파일을 전자우편으로부터 받을것이다. 이 파일들은 봉사기에 의하여 처리되는 모든 명령들과 그리고 가능한 목록들을 포함하고 있다. 사용자는 또한 어떤 목록에 대한 정보를 선택하여 볼수도 있다(info목록이름).

참고

목록을 탐색하는데 도움을 주는 두개의 Web싸이트가 있다. 첫번째 Web싸이트는 http://www.liszt.com이다. 이 싸이트는 90000개이상의 우편목록, 새소식그룹, IRC등록부들의 상세한 내용들을 포함하고 있다. 두번째 Web싸이트는 http://paml.net이다. 우와 같은 2개의 싸이트들은 목록을 쉽게 찾을수 있도록 탐색창문을 제공한다. 우편목록정보는 닉명ftp싸이트 ftp:\\ rtfm.mit.edu/pub/usenet/ners.answers.mail/mailing-lists에 보존되여 있다.

14.4 새소식그룹

인터네트가 나오기전에 UNIX사용자들은 USENET(사용자들의 망)라고 하는 토론회에 참석하였다. 여기서 USENET라고 할 때에는 사용자들이 특정한 주제에 대한 문제들을 제기함으로써 서로 정보를 교환하는 새소식그룹(newsgroup)전체를 이루는 말이다. 어떤 사람은 보통 제기된 문제들에 대한 대답을 주기 위하여 여기에 관심을 돌린다. USENET는 초기에 UUCP라는 통신규약을 사용하였으며 그후

TCP/IP로 바꾸었다. 인터네트에는 그 어떤 중심이 없이 10000개이상의 새소식그룹들이 있다.

USENET라는 용어는 새소식그룹을 의미하는 말인데 USENET라고 하는것보다 망새소식(Network 새소식) 간단히 새소식이라고 하는것이 보다 더 정확하다. 망새소식은 포구번호 119에서 NNTP(Network News Transfer Protocal)라는 통신규약을 사용한다.

새소식은 ISP들에 의하여 많은 새소식봉사기에 보존된다. 모든 새소식봉사기들은 규칙적인 간격으로 인터네트상의 다른 새소식봉사기로부터 새소식공급(newsfeed)을 받는다. 새소식의 량이 많아 지고 또 요구되는 디스크공간의 크기가 늘어 나는것으로 하여 자기 주위에 있는 ISP 혹은 망관리자는 모든 새소식그룹을 관리하지 못할것이다. 자기의 봉사기가 어떤 부분적인 새소식그룹에 대하여 주관하지 못한다면 이 경우에 사용자는 다른 봉사기에 접속할수 있다.

사용자는 망새소식을 리용하기전에 자기 콤퓨터에 새소식읽기의뢰기를 가지고 있어야 한다. 이것은 새소식봉사기로부터 새소식을 불러 들이고 머리부들과 내용들을 표시하는 프로그람이다. UNIX체계들은 nn, trn, tin과같은 문자기반의 새소식읽기도구들을 가지고 있다. 이러한 도구들은 다 자기의 우점과 결점을 가지고 있지만 그가운에서 tin은 사용하기 편리한 도구이다. 여기서는 도구 tin과 Netscape Messenger에 대하여 본다. Netscape Messenger는 GUI새소식의뢰기로서 봉사하는 우편처리기이다.

새소식을 받기 위해서는 하나 혹은 그이상의 새소식그룹에 가입해야 한다(새소식은 공개적이다). 이때 자기가 가입하려는 새소식그룹들의 이름을 새소식읽기프로그람(봉사기가 아니라)에 알려 주어야 한다. 전자우편과는 다르게 새소식은 다음의 두단계로서 호출된다.

- 먼저 **기사머리부**(article header)들이 전송되여 온다. 즉 머리부들을 보고 해당한 내용들을 선택적으로 볼수 있게 한다.
- 다음단계로서 그 내용 즉 기사(내용)들이 전송되여 온다.

기사들은 보통의 전자우편과 구별하기 위하여 개별적인 파일로서 보존된다. 새소식읽기프로그람은 가입한 모든 새소식그룹과 전송되여 온 기사머리부들의 경로를 보존한다. 망새소식에 의하여 발생되는 자료의 크기로 하여 새소식봉사기는 새로운 기사들을 위하여 낡은 기사들을 은퇴시켜야 한다. 이러한것으로 하여 통보문에 대한 내용의 호출이 불가능할수도 있다.

사용자는 기사나 혹은 통보문을 보내는것으로써 어떤 새소식그룹에 참가할수 있다. 응답으로서 어떤 사람은 자기자체의 어떤 기사를 보낼것이다. 여러개의 발송물들이 하나의 발송물로부터 분리되였다면 매개의 통보문들은 하나의 스레드(thread)를 이룬다. 사용자는 스레드단위로서 혹은 넌대순으로서 통보문들을 볼수 있다. 우편을 관리하는 Netscape Messenger를 리용하였다면 이미 이러한것들을 하였을것이다.

그러나 만일 NNTP를 리용하여 전화회선을 거쳐 자기의 ISP봉사기로부터 새소식을 불러 들인다면 사용자는 가입하려는 묶음들에 대하여 선택하여야 한다. 간단히 말하면 많은 묶음들과 많은 통보문들이 있다. 새소식그룹들에 대한 목록을 호출하는것은 상당한 시간이 걸린다. tin은 국부적인 봉사기와 함께 원만히 동작한다. 그러나 Netscape는 둘 다 적합하다.

주해

어떤 큰 기관에서 자기가 선택권한을 가진 봉사기에 대하여 새소식을 탐색하는것은 일반적이다. 이 봉사기는 고속회선을 통하여 매일 ISP봉사기로부터 내용들을 전송 받아야 한다. 자기 콤퓨터에 새소식을 불러 들인후부터는 그것을 쉽게 그리고 대단히 빠른 속도로 호출하여 볼수 있다.

14.4.1 이름짓기규칙

령역이름과 마찬가지로 새소식그룹들은 점으로 구별되는 이름들로서 구성되는데 차이점은 계층이 반대로 되여 있다는것이다. perl에 대한 문제들을 제기할수 있는 새소식그룹은 아래와 같은 형식을 가진다.

```
comp.lang.perl
```

점을 기준으로 할 때 왼쪽에 있는 이름은 오른쪽에 있는 이름보다 준위가 더 높다. comp(computers)는 **주묶음**이며 lang(language)은 그 밑준위에 있는 **보조묶음**이다. 즉 앞붙이 comp.lang을 가지는 새소식그룹들이 있으며 comp.아래준위에는 더 많은 새소식그룹들이 있다. 망에서 보게 되는 대부분 중요한 새소식그룹들은 USENET에서 찾을수 있다. 아래에 이러한 새소식그룹들을 보여 준다.

- biz: 사무활동
- comp: 콤퓨터분야 여기에는 두개의 보조묶음(comp.lang, comp.os)이 있다.
- news: USENET에 대한 모든것
- rec: 유희
- sci: 과학분야
- soc: 사회문화분야
- misc: 다방면적인 분야(그밖의 다른 분야에서 찾을수 없는 경우)
- talk: 최근 제기되는 많은 문제들에 대한 공개토론

최근년간 USENET이외에 여러개가 더 추가되였다. 이러한 새소식그룹이 보내는 문제들에는 많은 중복이 있게 된다. 그러므로 여러 새소식그룹들에 보내는 통보문들은 같은것일수도 있다.

14.4.2 새소식그룹규칙

새소식그룹에 대한 대중성은 FAQ(Frequently Asked Questions)로 인한것이다. 여기서 FAQ는 자원제공자들에 의하여 USENET게시판에서 진행된 편집물이다. 매 분야에는 그에 해당한 FAQ가 있다. 정해 진 질문에 대하여 새소식그룹에 기사를 보내기전에 그와 관계되는 FAQ를 보아야 한다. FAQ는 그와 관련된 새소식그룹과 news.answers에 월초마다 주기적으로 보내진다. FAQ는 닉명ftp싸이트 rtfm.mit.edu에 영구적으로 보존된다.

이러한 추세는 새소식그룹들에서 널리 보급되고 있으며 USENET에 그 기원을 두고 있다. 어떤 인상을 표현하는 ASCII문자들을 리용하면 대단히 효과적이다. 전자우편과 새소식통보문에는 :-)와 같은 표현이 자주 나타난다.

USENET는 foo,bar,foobar라는 용어의 출처에 대해서도 명확한 대답을 준다. 이 단어는 흔히 일반적인 파일로서 그리고 등록부이름으로서 리용된다. 이러한 단어들은 습관적으로 리용된다. 등록부를 만들기 위해서는 지령 mkdir foo를 혹은 파일을 복사하기 위하여 지령 cp foo bar를 리용한다. 이 책에서는 습관적으로 우와 같은 용어들을 리용한다.

14.4.3 .newsrc파일

새소식이 국부적인 봉사기로부터 읽혀 지든 NNTP를 사용한 인터네트로부터 읽혀지든 관계없이 tin

과 같은 문자기반의 새소식읽기프로그람들은 $HOME/.newsrc파일을 탐색하는것으로부터 자기의 동작을 시작한다. 이 파일의 매행에는 새소식그룹의 이름, 두점 (:), 감탄부호(!), 수묶음이 놓인다.

```
alt.animal.dolphins!                  비가입
alt.rec.movies: 1-561                  articles 1부터 561까지 읽는다
comp.os.linux.networking:1-721
comp.mail.sendmail: 1-250, 255-270, 275
rec.humor! 1-352               한때 가입되여 있었다
```

이 파일의 매개 행은 어떤 새소식그룹을 가리킨다. 새소식그룹이름과 수값묶음은 : 혹은 !로써 구분된다. 새소식그룹의 이름뒤에 구두점(:)이 놓이면 이것은 그러한 묶음에 가입된다는것을 의미하며 다음에 씌여 진 번호는 그에 해당한 내용을 참조하겠다는것을 의미한다. 수들은 반점으로써 구분되며 범위로써 지적할수도 있다.

일부 항목들은 :의 위치에 !가 놓여 있는것을 볼수 있다. 이것은 사용자가 묶음에서 탈퇴한다는것을 의미한다. 마지막행은 이전에 어떤 묶음에 가입하여 있었으며 352번까지의 내용들을 참조하였다는것을 나타낸다.

이것은 하나의 본문파일이기때문에 편집기를 리용하여 그것을 편집할수 있다. 즉 이 파일에 항목을 추가하는것으로써 새소식그룹에 가입할수 있으며 행을 지우는것으로써 묶음에서 탈퇴할수도 있다. 사용자는 또한 기사들에 표식(mark)을 붙일수 있다. 이것은 수값들을 리용하여 기사들을 읽기 위해서이다. 이러한 변화들은 tin, trn, nn와 같은 새소식읽기프로그람에 대하여 다 적용된다. 다만 이 파일을 읽지 않는것은 오직 Netscape이다.

14.5 새소식그룹읽기(tin)

가장 일반적인 문자기반의 망새소식읽기프로그람 tin에 대하여 보자. 아무러한 인수없이 그것을 호출하면 tin은 국부적인 새소식봉사기로부터 새소식을 불러 들인다. 다음 봉사기에 의하여 보존된 새소식그룹들의 능동목록과 .newsrc에 기록된 사용자목록을 비교한다. 만일 어떤 새로운 새소식그룹들이 존재한다면 tin은 그러한 목록들을 보여 주며 또 가입할수 있는 권한을 준다.

목록은 천여개가 넘는 묶음을 포함하는 대단히 큰 목록일수 있다. 이 경우 처음에 새소식을 불러 들인다면 목록이 화면에 표시되기전에 그 처리는 대단히 오랠것이다. 이러한 목록을 찾기 위하여 [pageup]과 [pagedown] 혹은 k와 j지령을 리용할수 있다. 자기가 어떤 묶음에 가입하려면 간단히 s를 누르시오. 같은 방법으로 묶음에서 탈퇴하려면 u를 리용하시오. 가입된 묶음들에 대한 목록은 .newsrc에 보존된다.

기사스레드(실지로는 통보문머리부)들을 보기 위해서는 묶음을 선택한후 [Enter]건을 눌러야 한다. Tin은 련관된 기사들에 대한 묶음을 하나의 스레드로서 결합하고 그러한 스레드들의 목록을 화면에 표시한다. 그림 14-2는 comp.os.sendmail 새소식그룹에 대하여 보여 준다.

목록에서 매 스레드에 대하여 왼쪽에는 계렬번호, 오른쪽에는 개발자이름이 제시된다. 또한 매개 스레드에 대해서 +기호는 아직 읽지 않은 내용들이 들어 있다는것을 암시한다. 10번 스레드는 11개의 기사를 가지고 있으며 12번 스레드는 내용이 없다. 보통 스레드서술의 뒤부분이 잘리우는 경우가 많다. 이 경우에 d를 누르면 내용이 확장된다. 본래상태로 돌아 가기 위해서는 다시 d를 누르시오.

참고

모든 초학자들은 묶음 news.announce.newusers에 가입해야 한다. 그것은 USENET기능에 관한 많은 정보를 가지고 있다. 즉 모든 사람들이 지켜야 할 행동규범과 새소식의 아래준위에 있는 묶음들에 대한 론의를 포함하고 있다. 단어 "answers"를 포함하고 있는 묶음에 가입하는것 역시 대단히 좋은 생각이다.

```
                com.mail.sendmail (114T(B) 157A OK OH R)            h=help

10  +  11  25 how to delete spam from mailq?          jose
11  +   2  46 pop accounts                            G. Roderick Singleto
12  +      53 Sendmail warning about "World Writable  G. Roderick Singleto
13  +   2  27 8.10.0.Beta6 SMTP Authenticating        Claus Assmann
14  +      89 Secondary mail                          Rob McMillin
15  +      69 HELP!  sendmail 8.9.3 on RH-6.1 thinks  Fisch
16  +   2  34 mailertable:how to send mail to relati  jlshan@wiscom.com.cn
17  +       6 forwarding e-mail                       amahcush@wt.net

<n>=set current to n, TAB=next unread, /=search pattern, ^K)ill/select,
a)uthor search, c)atchup, j=line down, k=line up, k=mark read, l)ist thread,
|=pipe, m)ail, o=print, q)uit, r=toggle all/unread, s)ave, t)ag, w=post
```

그림 14-2. tin으로서 새소식스레드목록을 보기

14.5.1 새소식기사의 처리

앞에서 본바와 같이 새소식머리부들이 먼저 내리적재된다. 사용자가 어떤 기사를 보지 못한 상태에서도 스레드로서 처리할수 있는 몇개의 지령들이 있다. 아래에는 이와 같은 tin의 지령들을 보여 준다.

o ----스레드를 표시한다.

s ----파일로서 그것을 보존한다.

w ----현재스레드에 련결되지 않은 새로운 통보문을 보낸다.

m ----친구에게 스레드를 보낸다.

K ---스레드에 읽기로서 표식(mark)을 붙인다.

읽기로서 스레드에 표식을 붙일 때 읽혀 지지 않은 스레드만을 보기 위하여 [tab]를 리용할수 있다. 읽어 진 기사들은 다음번에 tin에 의하여 표시되지 않도록 .news src파일에 기록된다.

만일 광고를 작성하려고 한다면 그 기사의 머리부가 사람들의 주의를 끌수 있도록 충분한 내용을 담고 있는가를 확인하시오. 대부분의 사람들은 이러한 머리부들을 보고 선택적으로 그 내용을 본다. 만일 통보문이 pppd problem와 같이 어지럽고 또 틀에 박힌 머리부라면 사람들은 흔히 그러한 통보문들을 보지 않을것이다. 대신에 pppd hangs in kernel 2.2와 같은것은 쉽게 무시되지 않을것이다.

tin은 몇개의 준위 즉 묶음준위, 스레드준위, 기사준위에서 조작을 진행한다. 그림 14-2는 스레드준위에서 본것이다. 어떤 준위에서 그 이전 준위로 이행하기 위해서는 q를 리용할수 있다. 만일 묶음준위에서 이러한 조작을 한다면 tin에서 완전히 탈퇴한다.

기사준위로 내려 가기 위해서는 목록에서 어떤 스레드를 선택하고 [enter]건을 누르시오. 10번 스레드에 대하여 이러한 조작을 한다면 그림 14-3과 같은 해당한 기사를 보여 줄것이다.

여기서는 스팸에 대한 처리와 관련한 Roger Marquis에 의하여 제기된 질문에 대하여 보여 준다. 류사한 선택항목들이 이 준위에서도 표시되며 두개의 추가된 항목들을 아래에 보여 준다.

```
Sun, 24 Oct 1999 20:41:25      comp.mail.sendmail      Thread   10  of  114
Lines 25           Re: how to delete spam from mailq?  10 Responses
jose <jose@biocserver.cwru.edu at Case Western Reserve University, Cleveland 0

Newsgroups:alt.spam,comp.os.linux.misc,comp.mail.sendmail,comp.security.unix

Roger  Marquis  wrote:

> Don't delete all your mail, just the spam.  The best way to do that
> is to find a unique character string used by the spam source,  say
> the original IP address,  then delete all messages containing that
> string.

then why not just set up a sendmail.cf config to do this? blck relaying,
accepting mail from known spammers (not relays but spammers), all of this
can be done very VERY easily in a sendmail.cf file. a simple m4 config
  ......
   <n>=set current to n, TAB=next unread, /=search pattern, ^K)ill/select,
   a)uthor search, B)ody search, c)atchup, f)ollowup, K=mark  read,
   |=pipe, m)ail, o=print, q)uit, r)eply mail, s)ave, t)ag, w=post
```

그림 14-3. tin으로 보여 준 기사

r-----기사(전자우편통보문)작성자에게 응답한다.

f-----새소식그룹통보문으로써 추적한다.

14.5.2 tin의 선택항목

tin은 인수로서 하나 혹은 그이상의 새소식그룹이름들을 가질수 있다.

```
tin comp.os.linux.networking
```

이것은 하나의 새소식그룹에 대한 기사를 받으며 조작에 있어서 비교적 속도가 빠르다. 또한 아래와 같은 방법으로서 새소식그룹의 완전한 적재를 피할수 있다.

tin -q	새로운 새소식그룹에 대하여 검사하지 않는다
tin -z	새로운 기사들이 있는 경우에만
tin -z rec.animals.wildlife	이 새소식그룹에 새로운 기사들이 있는 경우에만

사용자는 또한 tin을 리용하여 인터네트봉사기로부터 새소식을 읽을수 있다(실례로 ISP에 의하여 보존된 새소식). 이에 대해서는 간단히 새소식봉사기의 FQDN으로 변수 NNTPSERVER를 설정한다. 다음 export지령으로서 그 변수를 반출시키고 -r선택항목을 리용하여 tin을 기동하시오.

NNTPSERVER=news.vsnl.net.in	C쉘의 변수할당에 대해서는 8.11을 참고하시오
export NNTPSERVER	
tin -r	새소식을 읽는다

우의 모든것들은 보통의 전화회선을 리용하여 새소식봉사기와 련결할 때 속도가 대단히 떠진다는것을 제외하고는 이전과 같이 동작한다. 그러나 Netscape Messenger는 디스크에 기사머리부들을 영구적으

로 보존하므로 이러한 과제들에 대하여 대단히 유연하다. Netscape에 대해서는 다음절에서 취급한다.

참고

통보문들을 보낼수 있는가 그리고 받을수 있는가 하는것을 확인하기 위해서는 시험적으로 misc.test에 자기의 첫 통보문을 보내시오. 그러면 통보문이 주콤퓨터에 도착하는데 얼마나 오랜 시간이 걸리는가 하는것을 알수 있다.

14.6 망새소식을 위하여 Netscape Messenger를 리용하기

여기서는 새소식을 관리하고 Web의 열람을 제공하는 Netscape Communicator에 대하여 론의한다. 이 쏘프트웨어는 기본적으로 3개의 요소로 구성된다.

- Messenger: 모든 전자우편과 새소식그룹들을 관리한다. 이 요소는 제13장에서 전자우편을 론할 때 사용되였다. Netscape는 4.5판부터 Messenger에 새소식기능을 통합하였다.
- Navigator: World Wide Web를 열람할수 있으며 추가적으로 ftp와 telnet봉사를 제공한다.
- Composer: Web페지들을 만들기 위한 편집기능을 제공한다. 이 장에서는 오직 완성된 Web페지들을 전송 받기 위하여 이 요소를 리용한다.

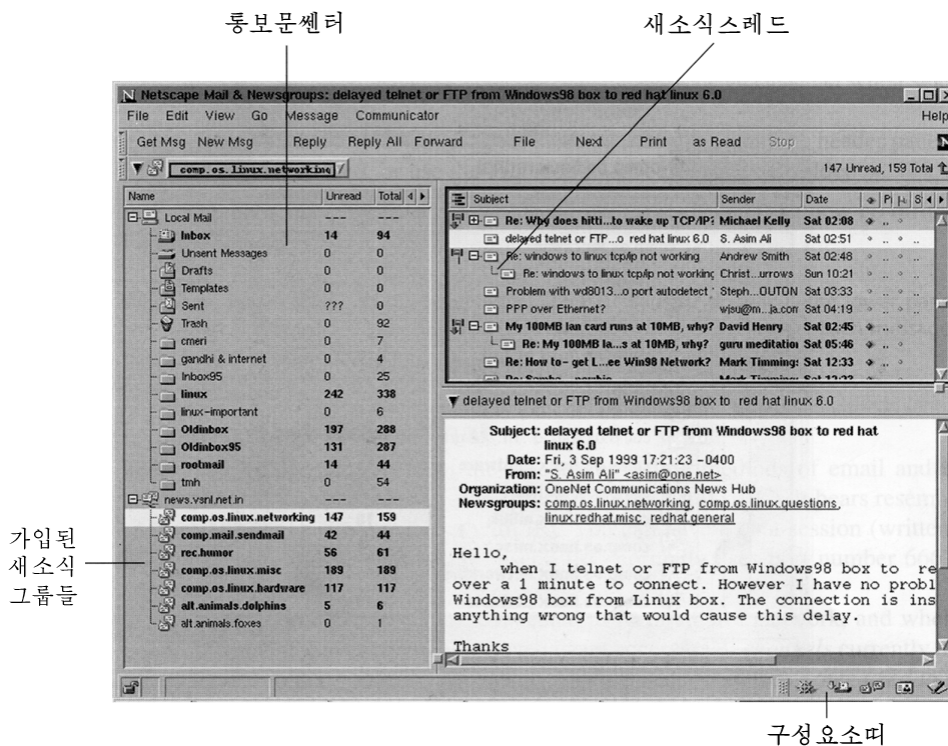



그림 14-4. Netscape Messenger에서의 새소식그룹들

통신처리기로서 이것들은 그 크기와 성능에 있어서 매우 빠른 시일내에 급속히 장성하였으며 현재 풍부한 X Window의뢰기로서 실행되고 있다. 그 크기로서는 12MB이상이다. 만일 하드웨어적인 담보가 없다면 기동시 대단히 지루한감을 줄것이다.

Xterm창문에서 netscape &지령으로써 communicator를 호출하시오. 13.7에서 이 지령은 Navigator 혹은 Messenger의 어느 하나를 보여 주었다. 어느 경우에도 창문의 차림표띠(그림 14-4)에는 Help차림표를 포함하는 6개 혹은 7개의 차림표항목들이 표시된다. 여기에는 File ,Edit ,View외에도 Communicator

라는 차림표항목이 있다. 이 차림표로부터 임의로 Communicator의 요소들을 선택할수 있다.

- Navigator: 열람기
- Messenger: 모든 우편들을 관리한다.
- Newsgroups: 망새소식을 읽거나 작성할수 있다.
- Address Book: 전자우편주소들을 보존한다.
- Composer: Web페지들을 만들기 위한 편집기

오른쪽밑부분에 5개의 아이콘을 포함하는 구성요소띠(Compoment Bar)가 있다. 우와 같은 5개의 모든 기능들이 구성요소띠에 함축되여 있다. 이것은 Communicator의 요소들을 보다 쉽게 호출하기 위해서 쓰인다.

전자우편과 새소식그룹은 기능적으로 볼 때 많은 공통점을 가지고 있다. Netscape는 이러한 공통점으로 하여 같은 사용자대면부를 제공한다. 사람들은 원격지에 있는 새소식봉사기로부터 새소식을 호출하기 위하여 우편을 관리하는 Messenger를 리용하므로 이에 대한 사용법을 잘 알아야 한다. 새소식을 봉사 받기 위해서는 아래와 같은 간단한 설정이 요구된다. 먼저 Edit>Preferences 를 선택하시오. 다음 새소식그룹 Server를 선택하고 거기에 자기의 새소식봉사기의 FQDN을 추가하시오. 이때 포구번호가 119로 설정되여 있는가를 확인하시오.

14.6.1 가입

새소식그룹에 가입하기 위하여 File>Subscribe를 선택하면 Messenger는 즉시 새소식봉사기와 련결되여 목록을 적재하기 시작한다. 만일 자기의 ISP봉사기로부터 전화회선을 리용하여 이러한 묶음들을 전송한다면 전체적으로 볼 때 그 처리는 반시간이상 걸릴것이다. 새소식그룹은 계층구조로서 창문에 나타난다(그림 14-5).

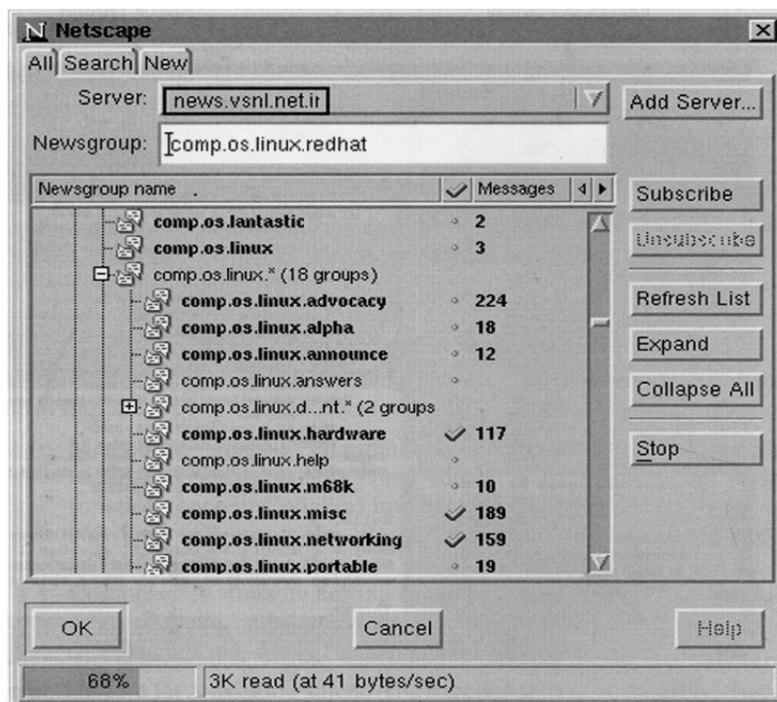



그림 14-5. Netscape Messenger에서 새소식그룹의 가입

창문에는 매개 새소식그룹에 대한 봉사기에서 유효한 통보문번호들이 표시된다. 묶음을 선택하고 Subscribe단추를 누르시오. 혹은 묶음에 대하여 마우스단추를 두번 찰칵하시오. 매우 큰 목록에서 어떤 묶음을 찾기 위해서는 그 이름을 입력하는것으로써 탐색기능을 리용하시오. 이것은 새소식그룹에 대한 완전한 이름이 정확히 알수 없는 경우에 리용한다.

어떤 묶음에 가입하면 통보문쎈터(그림 14-4)창문에는 봉사기이름(여기서는 news.vsnl.net.in)과 같은 이름을 가진 등록부아래로 새로운 등록부가 만들어 진다. 즉 가입된 새소식그룹들에 대하여 보조등록부가 만들어 진다. 사용자는 매개 새소식그룹을 마우스로 선택할수 있으며 통보문을 보내거나 받을수 있다.

참고

만일 새소식그룹의 이름을 알고 있다면 Web열람기는 거기에 가입하기 위한 가장 좋은 방법을 제공한다. 즉 URL문자렬에서 앞붙이로서 news를 리용해야 한다.

실례: `news:alt.animals.dolphins`

http://와는 다르게 여기서는 기호 //을 리용하지 않는다. [Enter]건을 누른후 통보문쎈터(Message Center)창문에 alt.animal.dolphins라는 이름을 가진 등록부가 만들어 진다.

14.6.2 통보문을 꺼내기

통보문의 꺼내기방법은 전자우편에서와 같다. 그러나 여기서는 전자우편에서와는 달리 먼저 새소식그룹을 선택해야 한다. 통보문쎈터창문에서 묶음을 선택한 다음 Get msg를 리용하시오. 이렇게 하면 머리부(스레드)들이 전송되여 오른쪽 창문에 나타난다. 이 시점에서 그의 해당한 내용들은 전송되지 않는다.

마우스로서 어떤 머리부를 선택하면 그의 내용은 일시적으로 전송되여 오른쪽아래창문에 나타난다. 이러한것들에 대하여 Tin에서와 같이 통보문들을 파일로서 보존할수 있고 읽기로서 그것에 표식을 붙일수도 있다. 또한 마우스로서 머리부들을 끌기하여 어떤 등록부에 옮길수 있다.

그러면 머리부들에 대한 어떤 묶음의 내용들을 어떻게 받겠는가? 내용들을 영구적으로 보존하는 조작은 좀 까다로우며 tin과 Netscape의 Windows판본이 UNIX Netscape보다 우월한 점이 바로 여기에 있다.

그러나 아래의 단계들을 따르면 보다 더 쉽게 할수 있다.

- 먼저 Local Mail폴더아래에 새소식그룹에 대한 새로운 폴더를 만든다(File>New Folder). Local Mail폴더는 이러한 조작을 하기전에 마우스에 의해서 찰칵되여 있어야 한다.
- 마우스로써 머리부창문안의 통보문들을 선택한다. 만일 머리부들을 하나의 련속적인 묶음으로서 선택하려면 첫번째 머리부를 찰칵한 다음 [shift]건을 누른 상태에서 마지막머리부를 찰칵하시오. 임의로 여러개의 머리부들을 선택하기 위해서는 ctrl건을 누른 상태에서 머리부들을 선택하시오.
- 다음 머리부창문으로부터 만들어 진 등록부에로 통보문들을 끌기한다. 이 시점에서 그 내용들은 완전히 전송되여 오며 통보문들을 비직결방식으로 볼수 있다.

참고

오른쪽아래창문에 머리부에 해당한 내용이 나타난 상태에서 File>Save As>File 을 선택하면 그 내용은 파일로서 보존된다. 이것은 매개의 통보문들을 개별적인 파일로서 관리할수 있다는것을 보여준다.

14.7 인터네트중계담화

오늘 사람들은 전자우편, 새소식그룹과 같은 비직결방식의 대화가 아니라 직결방식의 대화를 위주로 하고 있다. 이러한 봉사로서 인터네트중계담화(IRC:Internet Relay Chat)를 들수 있다. 이것은 UNIX의 talk지령과 류사하지만 여기서 말하는 IRC는 여러명의 사용자들과 실시간적으로 음성으로써가 아니라 본문으로써 중계대화를 진행할수 있는 봉사이다. IRC는 포구번호 6667을 리용하지만 일부 봉사기들은 6666을 리용한다.

IRC봉사기들의 묶음은 하나의 IRC망을 이룬다. 이러한 봉사기의 임의의것과 접속하면 그러한 망에서 모든 봉사기들에 의하여 관리되는 통로들에 대한 호출이 가능하다. IRC로서 대화를 진행하는 사용자들에 대하여 서로 같은 망에 있어야 하며 그렇다고 하여 같은 봉사기에 있어서는 안된다. 인터네트에는 이와 같은 몇개의 망(EFnet(Eris Free Net), Undernet, Dalnet)들이 있다.

IRC에서 모든 대화들은 통로에서 진행된다. 이러한 통로(channel)에 접속하기 위해서는 망에서 유일한 별명(nickname)을 리용해야 한다. 어떤 통로와 련결된 다음부터는 자기가 타자한 내용은 이 통로에 련결된 모든 사용자말단에 표시된다. 또한 통로들을 바꿀수 있으며 이 통로를 공유하여 파일로 보낼수 있다.

문자기반의 irc지령은 UNIX체계의 중요한 IRC의뢰기이다. 그것은 지령에 기초하고 있으며(표 14-3) man과 info문서가 없는것이 특징이다. 그러나 그것은 확장된 도움말기능을 제공한다.

표 14-3. irc에 의하여 사용되는 지령들

지 령	기 능
/list	통로들을 렬거한다
/list -min 5 -max 10	최소 5명 ,최대 10명의 사용자들에 의하여 공유되여 있는 통로들을 렬거한다
/list *98	통용기호에 맞는 통로들을 보여 준다
/server sname	FQDN sname을 가지고 있는 봉사기에 접속한다
/join #ch	#ch통로에 련결한다
/part #ch	#ch통로에서 탈퇴한다. 만일 #ch가 지적되지 않으면 현재통로에서 탈퇴한다
/msg usname mesg	usname으로 지적된 사람에게 mesg라는 통보문을 보낸다
/query usname mesg	usname으로 지적된 사람에게 mesg라는 통보문을 보낸다. /query지령을 리용하지 않아도 된다
Notify usname	usname이라는 대상이 망에 련결되였는가를 검사하기 위하여 이 지령을 사용한다
/whois usname	usname의 상세한 내용을 보여 준다
/away	림시적으로 통로에서 탈퇴한다
/invite usname #ch	통로 #ch로 usname이라는 대상을 끌어 들인다. 만일 통로가 지적되지 않는 경우에는 현재의 통로를 의미한다
/dcc chat usname	usname과 직접적인 접속을 진행한다
/dcc send usname flname	DCC를 통해서 usname으로 지적된 대상에flname이라는 파일을 보낸다
/dcc get flname	/dcc send 지령으로 보내 진 flname파일을 받는다
/quit	irc에서 탈퇴

14.7.1 통로에 대한 접속 및 렬거

어떤 IRC망에 접속하려면 별명을 리용해야 한다. 사용자는 별명을 리용하여 친구들에게 자기가 누구라는것을 알려 준다. 이 이름은 개별적인 망에서 유일해야 한다. Irc지령은 접속을 위하여 사용자의 별명과 봉사기의 이름을 요구한다.

```
$ irc wolfgang irc.cs.cmu.edu              별명은 wolfgang이다
.....
Ther are 5186 users and 44782 invisible on 42 servers
*** There are 170 operators online
*** 20683 channels have been formed
*** This server has 2990 clients and 1 servers connected
*** Current local users: 2990 Max: 4678
*** Currnet global users: 49968 Max: 53891
*** _
*** _
*** _
*** _
***  _          Carnegie Mellon University         Pittsburgh, PA
      .......
***  _             Eris-Free Net (EFNet) Internet Relay Chat (IRC)
***  _                     Ports 6666 - 6667 - 6668
      .......
```

우에서 보여 주는 부분은 20000개의 통로를 관리하는 EFnet망의 어떤 봉사기에 접속한 경우에 화면에 나타나는 출력자료이다. 이 상태에서 유표는 화면의 마지막행에 놓이게 된다. 여기에 모든 IRC지령(irc의 내부지령)을 입력하여 해당한 처리를 진행한다.

본문통보문과 irc지령들을 구분하기 위하여 irc내부지령들에 대하여 그앞에 기호 /을 리용한다. 실례로서 망우에서 현재 유효한 모든 통로들을 보기 위해서는 /list지령을 리용해야 한다.

```
/list
*** #utah       5      Nothing but good people in here.
*** #beatles    2      damn the man, save the Empire!
*** #polska     1
*** #linux      2      Support A Channel NOT Run By The Frankie BOT!!
*** #genimi     2      Let Kids Berkeley Kids…They Grow Up Too Fast!! :o/
*** #macintosh  3      PowerBook G3's Delayed, Fair-Poor Reviews Of Windoze 98,
*** #giggles    4      Have a Wonderfull Day!!! :)
    .......
```

여기서 기호 #는 공개된 통로라는것을 보여 준다. 우의 실례에서 두번째 렬은 통로에 대한 사용자수를 보여 주며 세번째 렬에는 알지 못할 본문이 표시된다. 조건에 맞는 통로만을 보려고 하는 경우 아래

와 같은 조건부를 가진 list지령을 리용한다.

```
/list -min 5                최소 5명의 사용자
/list -min 10 -max 15            10번부터 15번까지의 사용자
/list #Win98              단일행출력 - #win98은 렬거하지 않는다
/list *98             98을 포함하고 있는 모든 통로
```

어떤 원인으로 하여 접속이 자주 파괴될수 있다. 이 경우에는 irc에서 탈퇴하지 않고 /server지령으로 접속을 재실현한다.

참고

irc가 어떤 봉사기와 접속되지 않는 경우에는 -p선택항목으로서 다른 포구번호를 리용하시오. DALnet봉사기를 리용하기 위하여서는 다음과 같이 한다. irc -p 6662 wolfgang sodre.dal.net

14.7.2 통로에 대한 련결, 초청, 탈퇴

사용자는 /join지령을 리용하여 어떤 통로와 련결할수 있다. 대부분의 통로들은 공개되여 있기때문에 통로이름앞에 기호 #를 붙여야 한다.

```
/join #Win98                공개통로
```

만일 통로가 존재하지 않으면 이 지령에 의하여 통로가 새로 만들어 진다. 이 경우 사용자는 그 통로에 대한 조작자로 되며 일부 특수한 권한을 가진다. 통로가 존재한다면 사용자의 이름 즉 별명은 이 통로의 모든 사용자들에게 즉시 전해 진다.

```
*** wolfgang (henry@203.197.102.56) has joined channel #Win98
```

사용자가 건반을 통하여 입력한 내용은 모든 사용자에게 실시간적으로 보내 진다. 간단한 실례로서 다음과 같다.

```
<wolfgang> Does Windows 98 accept encrypted passwords?
<DeNIal> Sure it does
<wolfgang> That's why it doesen't connect to my Samba server
<DeNIaL> You have to make one important change
<DeNIaL> in the Windows registry.
<wolfgang> What's that?
```

통보문의 앞에 있는 이름이 바로 별명이다. 매개 행들이 너무 길지 않는가를 확인하고 입력된 즉시 [Enter]를 누르시오. 통보문은 [Enter]건을 눌러야만이 전송되므로 상대편쪽에서 본문을 전체로서가 아니라 대화식으로 볼수 있도록 한다.

또한 사용자는 자기의 현재통로 혹은 임의의 통로에 다른 사용자를 끌어 들일수 있다.

```
/invite Julie             현재의 통로에 julie를 끌어 들인다
/invite Julie #y2k_help             통로 #y2k_help로 julie를 끌어 들인다
```

이 지령을 처리하면 julie는 자기의 화면에서 다음과 같은 통보문을 보게 된다.

```
*** wolfgang invites you to join #y2k_help
```

이 상태에서 그는 지령 /join으로써 통로에 련결할수 있다. 사용자는 /quit와 /bye지령으로서 irc에서 탈퇴할수 있다. 그러나 이에 앞서 통로를 끊어야 하며 이를 위해서는 /part지령을 리용해야 한다. 또한 현재통로뿐만이 아니라 그 어떤 특별한 통로에 대해서도 끊을수 있다.

```
/part                IRC에서 탈퇴한다. /leave지령을 리용할수도 있다
/part #Win98                   지적된 통로만을 끊는다
```

참고

사용자는 통로에서 탈퇴하여 새로운 통로에 접속하지 않고도 지령/msg nickname message을 리용하여 개별적으로 사람들과 대화를 진행할수 있다. 이 지령을 리용하면 통보문은 수신콤퓨터에만 표시된다. 매 시각 이 지령의 사용을 피하기 위하여 /query 지령을 리용한다. 일단 그것이 사용되면 모든 련속적인 통보문들은 개별적인 화면에만 나타난다.

14.7.3 직접련결과 파일교환(DCC)

사용자는 망우에서 특히는 IRC봉사기에서 자주 질문을 받게 된다. irc의 **직접의뢰기접속**(Direct Client Connection:DCC)기능을 리용하여 사용자는 개별적으로 어떤 사람과 직접 통신할수 있다. 이 경우에 사용자는 통로에 접속하거나 또 IRC봉사기를 공유할 필요가 없다. 이러한 특징은 파일을 교환할 때 대단히 유용하다.

DCC는 지령 /dcc로써 호출된다. 지령은 몇개의 예약된 인수들을 가지고 있다. monty와 DCC대화를 진행하기 위하여서는 chat인수를 리용해야 한다.

```
/dcc chat monty                chat는지령 /dcc의 인수이다
```

사용자는 DCC기능을 리용하여 본문 혹은 2진파일을 보낼수 있다. 파일의 전송은 /dcc지령의 인수 send와 get를 리용하여 두 단계로서 진행된다. 발송자는 다음과 같은 지령으로 전송을 시작한다.

```
/dcc send monty penguin.gif
```

새소식은 monty에게로 전달되며 반대로 monty는 auto-get파일선택항목이 리용되지 않는다면 파일을 받기 위하여 지령 /dcc get를 호출해야 한다.

```
/dcc get penguin.gif                파일인 경우에만 그 이름을 놓을수 있다
```

전송은 직접적으로 진행되며 전자우편첨부물(attachment)과 비교해 볼 때 더 적은 비용이 든다. 더 중요하게는 동시적이라는것이다. 우의 DCC대화는 지령 /dcc close chat monty로써 닫을수 있다.

DCC기능은 IRC의 강력한 특징이다. 이 방법은 IP주소를 리용하지 않고도 파일을 전송할수 있는 가장 편리한 방법이며 또 보안기능이 대단히 강하다. 즉 비밀토론을 진행할 때 아주 유용하다. 그것은 닉명 ftp에 대한 혼란과 전자우편첨부물들에 대한 제한사항들을 극복한다. IRC를 통하여 5MB의 파일을 보내는것은 아주 간단하다. 전자우편으로서 이 파일을 전송하려 한다면 ISP는 5MB크기의 우편함을 제공하지 않는다.

14.8 World Wide Web

인터네트의 내용이 확대됨에 따라(그리고 더욱 무질서화되였다.) 사용자들은 어떤 주제에 대한 정보를 검색하기가 이전보다 더 어려워 진다는것을 알게 되였다. 이 과정에 그들은 telnet와 ftp싸이트 ,우편목록, 새소식그룹들을 하나의 덩어리로 묶어 보려고 하였다. 무엇보다 중요한것은 검색속도를 높이기 위하여 어떻게 정보를 조직하겠는가 하는것이였다. 이 시기 사람들은 간단한 건누룸조작으로써 정확한 위

치에로 이동할수 있는 하나의 단일도구도 요구하였다.

다음세대도구들인 Archie와 Gopher는 여러가지 시도를 진행하였다. 그러나 1991년에 이르러 스위스 CERN의 팀 베르나슈 리(Tim berners-Lee)에 의하여 문서를 호상 련결하는 수단이 나왔다. 이 체계에서 문서는 원격지에 있는 다른 콤퓨터는 물론 거기에 있는 문서에 대한 읽기프로그람을 가리키는 충분한 정보도 포함하고 있다. 그는 문서들이 호상 련결된 이러한 망을 **Web**라고 불렀다. 문서열람기(document viewer, browser)가 그의 짝패로 등장함으로써 Web는 세계적인 망 즉 **World Wide Web**로 발전하였다.

인터네트와 마찬가지로 Web는 의뢰기-봉사기모형으로서 작업하며 Web의 접근(의뢰기)도구를 **열람기**(browser)라고 부른다. Web열람기는 Web봉사기측의 문서와 화상들을 불러 들이며 문서는 자체에 제공되여 있는 서식화명령들을 리용하여 문서들을 서식화한다. 열람기는 GIF와 JPEG형식의 화상들을 표시한다. 그러한 형식이 아닌 화상들에 대하여 열람기는 방조응용프로그람을 호출한다. 여기서 방조응용프로그람은 어떤 외부응용프로그람이다.

자원은 싸이트의 FQDN과 파일의 경로이름을 결합하는 주소지정형식인 **URL**(Uniform Resource Locator)에 의하여 서술된다. 즉 사용자들은 URL로서 Web싸이트에 접근한다. URL의 가장 일반적인 형식은 다음과 같다.

`http://www.who.org`　　　　세계보건기구

Web에서 자원단위는 하나의 **Web페지**이며 일반적으로 두세개의 본문-그림들로 이루어 진다. **하이퍼본문표식달기언어**(HTML: Hypertext Markup Language)는 이러한 Web페지들을 서술하기 위한 일종의 언어이다. HTML문서는 임의의 조작체계를 리용하는 임의의 장치에서 열람기에 의하여 표시된다.

문서에서 본문은 다른 Web페지에 대한 **하이퍼련결**(hypertext link 혹은 hyperlink)들을 가지고 있다. 이러한 련결을 리용하여 같은 문서에서 서로 다른 위치, 같은 봉사기에서 서로 다른 문서, 그리고 인터네트상의 임의의 페지들을 볼수 있다. 또한 본문은 화상들과 련결될수 있으며 화상 그자체는 다른 화상들과도 련결될수 있다. 이러한 련결은 간단한 건누름이나 마우스누름으로써 시작된다.

Web는 련결된 모든 자원들을 꺼내기 위하여 포구번호 80에 **하이퍼본문전송규약**(HTTP: Hypertext Transfer Protocol)을 리용한다. 이러한 원인으로 하여 Web봉사기들은 HTTP봉사기라고도 부르며 열람기들은 HTTP의뢰기라고도 부른다. 현재의 열람기는 대부분의 기초적인 봉사를 다 제공하기때문에 이것으로써 이전의 인터네트의 많은것들을 볼수 있다. 즉 이 열람기로부터 URL의 앞붙이로서 ftp://와 telnet://를 리용하여 ftp(11.7)와 telnet(11.5)를 사용하였다. 이때부터 개별적인 ftp와 telnet의뢰기가 필요 없게 되였다. 전체적으로 볼 때 열람기는 인터네트에서 한번의 조작으로 모든것을 다하는 도구(one stop shop)이다. 이 절에서 URL, HTML, HTTP라는 용어가 나오는데 이러한 용어들에 대해서는 다음 절에서 본다.

14.9 Web열람기 Netscape Navigator의 리용

Web열람기는 HTTP의뢰기라고 말할수 있다. 이러한 HTTP의뢰기들은 HTTP봉사기로부터 문서를 불러 들이고 그것을 화면에 표시한다. 초기에 Web는 본문만으로 이루어 져 있었기때문에 본문열람기라고도 불렀다. lynx는 가장 널리 알려 진 본문열람기이며 문서검색속도가 대단히 빠르다. 그러나 사람들은 이것이 도형들을 관리할수 없는것으로 하여 잘 리용하지 않는다.

후에 Web에 도형처리기능을 도입하였으며 오늘날 사람들은 X Window체계에서 실행되는 도형열람

기들을 리용하고 있다. Netscape는 UNIX체계에서의 표준열람기이다. 종류에 관계없이 모든 열람기들은 다음과 같은 특징들을 가진다.

- 문서에 대하여 앞뒤로 번질수 있다.
- HTML파일(그리고 도형)들을 국부콤퓨터에 보관할수 있다.
- URL을 입력하지 않고도 후에 꺼낼수 있도록 중요한 URL들에 서표(bookmark)를 줄수 있다.
- FTP와 Telnet와 같은 다른 응용프로그람규약들을 지원한다.
- 자동적으로 자기가 관리할수 없는 파일형식에 대하여 방조응용프로그람과 특수한 쏘프트웨어(plugin)를 호출한다.

그림 14-6에 Navigator창문을 보여 준다.

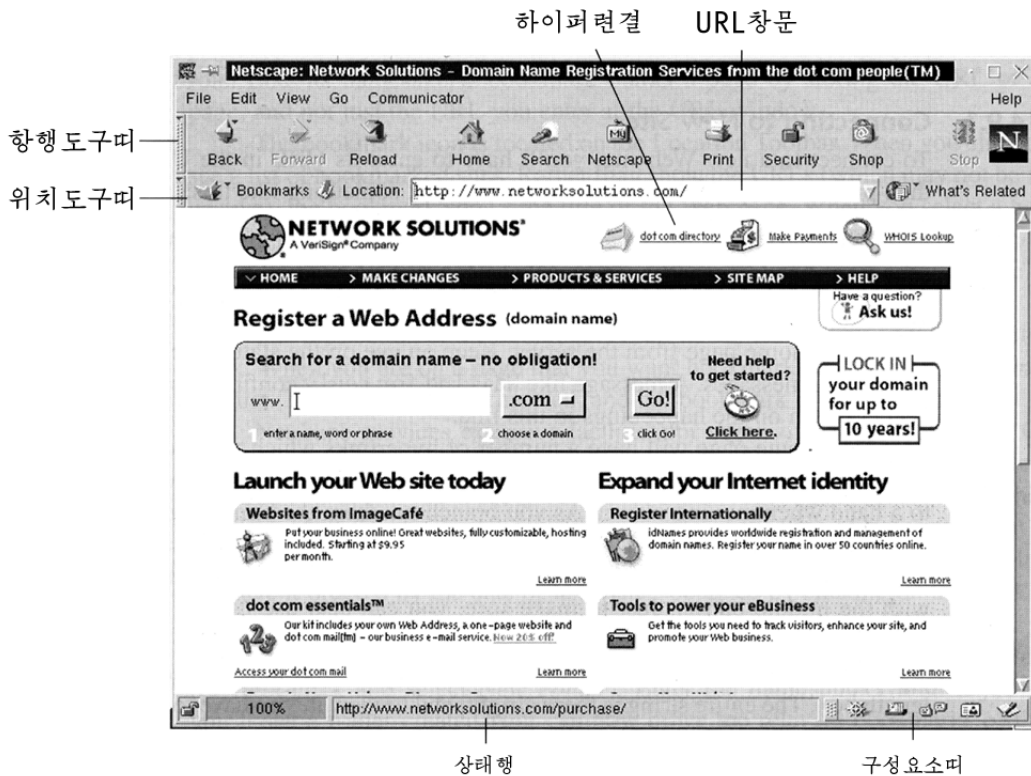



그림 14-6. Netscape Navigater로 본 Web페지

Messenger를 리용할 때와 같이 차림표띠로부터 Communicator>Navgator를 선택하시오. 기본차림표아래에 표식이 붙은 아이콘들로 이루어 진 3개의 도구띠가 있다. Navigator도구띠는 다음과 같은 단추들을 포함하고 있다.

- Back와 Forward: 열람기는 사용자가 선택한 모든 페지들에 대한 URL들을 기억하고 있다. 그러므로 이 단추를 리용하여 이전 혹은 이후 상태의 페지로 쉽게 절환할수 있다. 즉 URL을 다시 기입하는 조작이 없이도 한 대화에서 보았던 모든 문서들을 재표시할수 있다.
- Reload: 봉사기로부터 마지막페지를 불러 들이기 위하여 이 단추를 리용한다. 페지들은 대체로 동적으로 만들어 지며 마지막정보를 얻기 위하여서는 이 단추를 눌러야 한다.

위치도구띠(Location Toolbar)는 URL창문과 Bookmarks아이콘을 포함하고 있다. 서표에 대해서는 이 장의 마지막부분에서 론한다. 여기서는 Netscape싸이트의 다양한 자원들과의 련결을 가지고 있는 개

인용도구띠(Personal Toolbar)에 대해서는 론하지 않는다.

참고

사용자는 개인용도구띠를 제거하여 내용표시령력을 늘일수 있다. 차림표띠로부터 View를 선택한 다음 Personal Toolbar를 비선택으로 한다. 또한 사용자는 도구띠의 도형아이콘들을 제거하여 표시령역을 더 늘일수 있다. 도구띠를 본문만으로 표시하기 위하여서는 Edit>peferences> Appearance를 선택하고 Text Only단일선택단추에 검사표식을 하시오.

Web싸이트들에 접속할 때 아래의 상태행을 보시오. 어떤 하이퍼련결 혹은 아이콘을 누르면 이 행에는 현재 Web열람기가 무엇을 하고 있는가 하는 통보문이 표시된다. 여기에 표시되는 통보문들은 다음과 같다.

- 싸이트에 련결될 때와 Web페지가 호출되기전에 첫번째 통보문이 여기에 나타난다.
- 하이퍼련결우로 마우스를 움직일 때 그에 해당한 URL을 보여 준다.
- 자원이 전송되여 오는 속도와 퍼센트를 보여 준다.
- 때때로 싸이트의 IP주소를 보여 준다.
- Document Done통보문은 페지가 완전히 전송되였다는것을 보여 준다.

Navigator가 기동되면 열람기창문에는 구성방식에 의존하는 내용이 표시된다. 기정으로서 열람기는 Netscape의 싸이트 home.netscape.com에 대한 홈페지에 련결한다. **홈페지**는 URL창문에 싸이트의 FQDN을 입력할 때 나타나는 첫 페지이다. 즉 소개페지라고 말할수 있다. 사용자는 Edit>Preferences> Navigator를 선택하여 이러한 기정상태를 변경시킬수 있다. 즉 빈 페지를 표시한다거나 마지막으로 본 싸이트를 표시하려고 할 때 우와 같은 조작을 한다.

주해

모든 통신기능들은 Edit차림표(Edit>Preferences)로부터 구성된다. 항목들은 기능단위로 구별되여 있으며 파라메터구성에서 제기될것은 하나도 없다.

14.9.1 새로운 싸이트에 접속하기

새로운 Web싸이트에 접속하기 위해서는 URL창문에 해당한 URL을 입력하고 [Enter]건을 눌러야 한다. URL은 일반적으로 아래와 같은 형식으로 되여 있다.

```
http://www.nasdaq.com                NASDAQ에 대한 Web사이트
```

앞붙이 http://는 특별히 요구되지 않지만 Web싸이트라는것을 지적하기 위하여 일반적으로 쓰인다. 열람기는 싸이트에 련결하여 봉사기로부터 홈페지를 불러 들인다. 이때 상태행에 나타나는 통보문을 보시오. Waiting for reply라는 통보문이 제시되는데 이것은 잠시 기다려야 한다는 의미이다.

홈페지는 몇개의 하이퍼련결들을 가지고 있으며 이러한것들은 밑선이 그어 진 본문을 보고 쉽게 알수 있다. 또한 화상들우로 마우스가 움직일 때 손그림지시자로 변하면 이것은 하이퍼련결로서 동작할수 있다는것을 보여 준다. 사용자는 하이퍼련결들을 누르는것으로써 홈페지로부터 해당한 내용을 볼수 있으며 자동적으로 URL창문이 갱신된다. 사용자는 간단히 그것을 선택하여 복사할수 있으며 또 임의의 위치에 붙이기할수 있다. 이러한 조작은 [Alt-c],[Alt-v]로써도 실현할수 있다.

때때로 문자렬이 너무 긴것으로 하여 URL창문에 다 표시되지 않는 경우가 있다. 이 경우에 여기에 유표를 놓고 방향건을 리용하여 볼수 있다. 문자렬에서 ?, +, =와 같은 문자를 자주 보게 되는데 URL들은 지령과 마찬가지로 인수 즉 파라메터를 가지고 있다.

사용자는 URL창문에 자기가 알고 있는 URL들을 입력하여 해당한 Web싸이트의 내용을 볼수 있다. 이때 열람기는 이러한 URL들을 기억해 둔다. 때문에 자기가 지금까지 탐색해 온 내용을 반대로 추적하여 이미 보았던 문서들을 다시 볼수 있다. 우와 같은 경우는 망우에 접속되여 있는것을 전제로 하며 Back와 Forward단추로서 실현한다. Back단추를 계속 누르면 마지막에는 처음에 보았던 문서가 화면에 나타난다.

14.9.2 URL리력과 서표

대화중에 어떤 문서를 추적하기 위하여 Back와 Forward기능을 리용하는 방법은 효과적인것이 못된다. 즉 하나의 문서를 보기 위하여 지금까지 본 모든 페지들을 다 훑어 볼수는 없다. 이 경우 두가지 편리한 방법이 있다.

- Back단추를 마우스의 왼쪽단추로 눌러 나타난 튀여나오기목록으로부터 싸이트를 선택한다. 이 목록은 열람기를 탈퇴하기전까지는 기억기에 남아 있는다.
- 사용자가 보았던 싸이트들에 대한 목록은 URL창문의 내리펼침칸에 나타난다. 여기서 싸이트들을 선택하여 볼수 있다. 이것들은 9일동안 존재하지만 사용자는 자기의 편리에 맞게 그 날자를 변경시킬수 있다(Preferences>Navigator> History).

URL창문에 대한 리력기능에는 한가지 문제점이 있다. 열람기는 URL창문에 련결로서 보게 되는 페지들에 대한 URL이 아니라 싸이트에 대한 URL을 보여 준다. 여기서 **서표**(bookmark)는 중요한 역할을 한다. 서표는 URL창문에 입력한 URL이 아니라 임의의 Web페지에 대한 URL을 보존하는것으로써 페지에 표식을 붙인다.

서표아이콘은 위치도구띠에 있다. 그것을 누르면 서표가 붙은 싸이트들의 목록이 현시된다. 서표는 Web페지에 대한 URL이 아니라 그의 제목을 보여 준다. 그것을 선택하면 사용자는 즉시 그에 맞는 싸이트와 자원에 련결된다. 사용자가 가질수 있는 서표의 수에는 제한이 없으며 홈페지에 나타나는 모든 련결들에 서표를 줄수 있다.

서표의 내리펼침목록의 웃부분에는 그것을 추가하고 편집할수 있게 하는 항목들이 있으며 서표를 정리하여 보기 위한 항목도 있다. 서표를 주려는 페지에 대하여 이 목록의 Add Bookmark항목을 선택하시오. 선택된 페지에는 서표가 붙어 목록의 끝에 추가된다.

Netscape는 서표들을 구성하기 위한 확장된 기능을 제공한다. 몇개의 서표들을 가지고 있는 경우 이러한것들을 적당한 등록부에 배치하면 보기에도 대단히 편리하다. Edit Bookmark를 선택하면 그림 14-7에서와 같이 계층적인 방식으로 구성된 서표들에 대한 창문을 보게 될것이다. Messenger에서와 같이 이 창의 title행을 누르는것으로써 어떤 순서로 목록의 내용을 정렬시킬수 있다. 실례로서 Last Visited라는 title을 누르면 시간에 따라 목록의 내용이 정렬된다.

Books and Magazines라는 등록부를 만들기 위해서는 File>New Folder를 선택하시오. 혹은 오른쪽단추를 눌러 거기서 New Folder를 선택할수 있다.

이름 Books and Magazines를 입력하고 [Enter]를 누르면 창문에서 등록부가 만들어 진것을 볼수 있다. 사용자는 두가지 방법으로 이러한 등록부에 서표들을 옮길수 있다. 첫번째 방법은 그것을 하나하나 끌어다 놓는것이다. 또 다른 방법은 [Ctrl]건 혹은 [Shift]건으로 그것들을 묶음으로서 선택하고 [Alt-x]로 자른 다음 [Alt-v]로 붙인다.

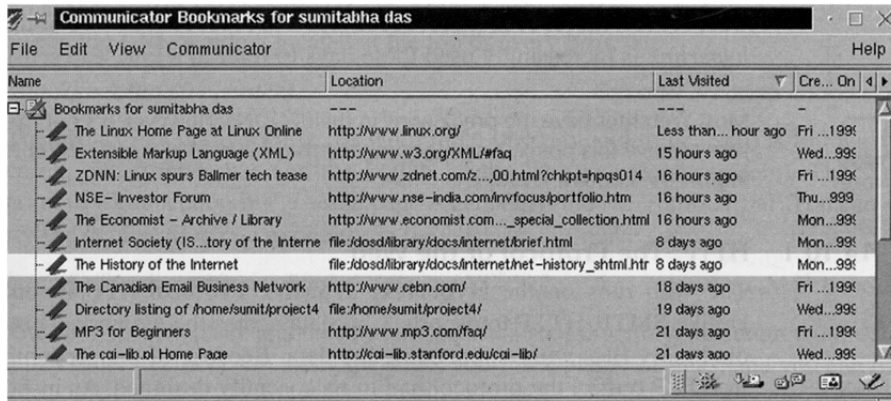



그림 14-7. 서표창문

참고

서표를 리용하지 않고도 이전에 보았던 싸이트들을 호출하기 위한 또 다른 방법이 있다. Netscape는 모든 URL들에 대한 상세한 내용들을 보존하며 기정으로 9일간 보관한다. 만일 하나의 싸이트에 대하여 서로 다른 10개의 페지들을 보았다면 Netscape는 모든 10개의 URL들을 보존한다. 사용자는 Communicator>tools>History를 리용하여 다른 창문에 보존된 모든 URL들을 볼수 있다. 이 창문은 처음 본 시간, 마지막으로 본 시간, 보관날자 그리고 몇번 보았는가 하는 총수를 보여 준다. URL목록에서 임의의 항목을 두번찰칵하면 그에 해당한 문서를 호출한다. 이것은 아주 쓸모 있는 기능이며 기억하기 힘든 URL에 해당한 페지를 찾기위한 편리한 방법이다.

14.10 하이퍼본문과 HTTP, URL

Web문서를 어떻게 구성하는가 하는것을 리해하기 위하여 간단한 실례를 보기로 하자. 대체로 사용자들은 어떤 책을 읽을 때 처음부터 마지막까지 차례로 읽는다. 그러나 그것은 책의 내용을 파악하는데서 가장 위력한 방법은 아니다. 책을 읽는 과정에 리해되지 않는 부분에 맞다들리면 이전 장에서 서술된 내용들과 다른 책의 내용들을 참고하여야 한다.

Web 역시 이와 비슷하다. 여기서는 하이퍼본문이라는것을 리용하는데 이 단어는 Web의 HTTP와 HTML(Web에서 리용하는 통신규약과 언어)에서 보게 되는 단어이다. 이것은 비선형방식으로 읽을수 있도록 본문을 구성하기 위한 체계이다. Web페지들은 다른 페지들에 대한 참조들을 포함하고 있으며 이러한것들을 **하이퍼본문련결**(간단히 **하이퍼련결**)이라고 한다. 이러한 련결들은 아래와 같은 방법으로 HTML문서의 주본문에 매몰된다.

```
<A HREF="http://www.sonu.com/docs/email.html">The email debate</A>
```

HTML에 대해서는 후에 취급하기로 하고 현재는 우의 실례에서 A와 HREF라는 두 단어에 대하여 리해하는것이 보다 중요하다. 열람기는 이러한 행을 만나면 "The email debate"라는 본문에 밑선을 그으며 또 그것을 색으로 표시한다. 그러면 이러한 밑선 친 본문이 바로 하이퍼본문련결이다. 이 부분에서 마우스를 누르면 이 련결(www.sonu.com)에서 지적된 Web봉사기로부터 문서를 꺼내여 열람기창문에 그것을 표시한다. =의 오른쪽에 있는 부분은 인터네트싸이트에 대한 URL이다. 이렇게 불러 들인 문서는 역시 다른 문서와 화상들에 대한 참조인 A HREF끄리표(tag)를 가지고 있다. 보다 중요한것은 화상들도 다른 자원들을 지적할수 있다는것이다. 이 원인으로 하여 최근에 하이퍼련결(hyperlink)이라는 용어는 련결(link)이라는 말을 대표하고 있다.

주해 대부분의 Web싸이트는 자기의 FQDM에 앞붙이 www를 가지고 있다. 그러나 이러한 앞붙이는 모든 Web에 대하여 붙는것이 아니다. 어떤 기업체는 그에 해당한 여러개의 앞붙이를 가질수 있거나 전혀 가지지 않을수도 있다.

14.10.1 Web의 통신규약 HTTP

Web는 포구번호 80으로서 **하이퍼본문전송규약**(HTTP)에서 실행된다. SMTP와는 달리 HTTP는 망을 통하여 8bit자료를 전송한다. 이것은 자료를 코드화함이 없이 2진파일들을 조종할수 있다는것을 의미한다. HTTP에서는 도형을 처리할수 있다는 우월성으로 하여 이 규약은 특별히 설계되여야 하였다. ftp에서와 같이 접속은 아래와 같은 단계로 진행된다.

- 의뢰기는 URL에서 지적된 FQDN에 해당한 봉사기와 련결하며 포구번호 80으로 접속을 개시한다.
- 다음 의뢰기는 봉사를 받기 위하여 Web봉사기에 요청한다. 이렇게 되면 사용자는 봉사기로부터 자료를 받을수 있으며 또 보낼수도 있다. 요청은 의뢰기에 의하여 보내진 자료에 따르는 **요청머리부**(request header)로 이루어 진다.
- 봉사기는 **응답머리부**(response header)로 이루어 진 응답을 보낸다.
- 봉사기는 제기될 요청을 기다리며 마지막에 접속을 끝낸다.

우의 단계는 HTTP 1.1을 실행시키는 Web봉사기에 대한 동작형태이다. HTTP 1.0에서 모든 자원들은 서로 다른 접속을 통하여 Web봉사기로부터 호출된다(일부 봉사기들은 단일접속을 리용하여 여러 자원들을 호출할수 있게 한다). 이것은 하나의 Web페지가 5개의 도형파일을 가지고 있을 때 그것을 전송하기 위해서는 6개의 접속이 필요하다는것을 의미한다. HTTP 1.1은 기정으로 **영구접속**(persistent connections)을 지원한다. 이것은 단일한 접속으로써 일감을 처리한다는것을 의미한다. 통신규약은 어느 경우이든 관계없이 매 접속이 다른 접속에 대해서 전혀 모르는(접속들이 련속적으로 진행된다 할지라도) **무상태규약**(stateless)이다.

어떤 봉사기에 보내는 요청머리부는 자료를 보내기 위하여 사용되는 메쏘드(method)를 가지고 있다. HTTP는 봉사기로부터 어떤 문서를 얻기 위하여 get메쏘드를 리용한다. 또한 양식자료(form data)가 열람기로부터 봉사기로 보내질 때 post메써드가 자주 리용된다. 봉사기는 머리부에 따르는 자료가 또 한번의 처리를 위하여 **관문프로그람**(gateway program)에 전송되여야 한다는것을 <Form>끄리표로부터 인식한다. get, post메써드는 20.20.3에서 론의한다. CGI프로그람작성자들은 이러한 메쏘드들을 리용하여 자료를 보낼 때 그것이 어떤 형식으로 구조화되여야 하는가를 알고 있어야 한다.

봉사기의 응답머리부는 전송되여 오는 자료의 형을 서술한다. MIME의 내용형(13.9)이 바로 여기에 해당된다. HTML문서를 보내기 위하여서는 응답머리부의 내용형마당은 text/html로 되여 있어야 한다. 만일 봉사기가 관문프로그람에 양식자료를 넘긴다면 머리부는 봉사기에 의해서가 아니라 관문프로그람에 의하여 발생되여야 한다. perl에서 CGI프로그람을 작성하려 한다면 자기의 프로그람을 통하여 그것들을 발생시켜야 하므로 이러한 머리부마당들에 대한 정확한 형식을 알고 있어야 한다.

14.10.2 URL의 구조해석

Web싸이트에 접근하기 위하여 사용자는 열람기의 웃부분에 위치하고 있는 URL창문에 싸이트에 해당한 URL을 입력하여야 한다.

```
http://java.sum.com[Enter]                FQDN대신에 IP주소를 입력할수 있다
```

URL은 단어 http와 두점(:), 두개의 사선(//), Web싸이트에 대한 FQDN(java.s um.com)으로 이루어 진다. 접속의 첫 시작점은 홈페지이며 일반적으로 index.html파일이다. 사용자는 URL창문에 이 기정파일이름을 직접 지정하여 볼수도 있다.

```
http://java.sun.com/index.html              index.html은 기정파일이름이다
```

파일 /index.html은 봉사기의 뿌리등록부(체계의 뿌리등록부와는 다르다.)에 있다. 홈페지에는 많은 련결들이 있으며 그것을 찰칵하여 그에 해당한 내용을 볼수 있다.

또한 사용자는 문서의 절대경로를 지적할수도 있다.

```
http://java.sun.com/docs/books/tutorial/index.html
```

여기서 2중사선기호 //은 반드시 있어야 하며 단일사선기호 /은 등록부를 지적할 때 리용한다.

```
http://java.sun.com/docs/books/tutorial/              tutorial은 등록부이다
```

URL은 간단한것으로부터 복잡한것에 이르기까지 확장된다. URL은 문법적으로 다음과 같은 3가지 규칙들의 결합이다.

- 자원을 전송하기 위하여 리용되는 통신규약(http://)
- 주콤퓨터에 대한 FQDN(java.sun.com)
- 파일의 경로 (/docs/books/tutorial/index.html)

략어 URL의 확장(Uniform Resource Locator)에서 두개의 단어 Uniform과 Resource에는 일정한 의미가 있다. Web봉사기로부터 불러 들인 자료는 하나의 자원이라고도 말할수 있다. 즉 봉사기는 파일을 보내는것이 없이 동적인 HTML내용을 만들수 있다. Uniform이라는 용어에 대하여 말한다면 HTTP 통신규약에서는 앞붙이 ftp://와 telnet://로서 다른 통신규약을 사용할수 있으므로 주소화방법은 하나로서 일정하다. 아래의 실례는 열람기에 의하여 지원되는 서로 다른 통신규약의 사용을 보여 준다.

- ftp://ftp.tucows.com
  이러한 URL을 리용할 열람기에서 닉명ftp를 사용하면 파일들을 목록화한 어떤 등록부를 보여 것이다.
- telnet://sasolution.com
  X Window체계는 telnet를 호출하기 위하여 간단히 xterm의뢰기를 리용한다.
- Gopher://manuel.brad.ac.uk/11/.faq/.unix
  Gopher싸이트에 접속한다.
- Malto:thathc@netscape.com
  이와 같은 련결을 가지고 있는 본문에 대하여 마우스를 누르면 전자우편을 보낼 때 리용되는 Netscape Messenger의 구성창문을 호출한다.
- File:///home/henry/download/penguin.gif
  국부콤퓨터에서 파일들을 보기 위하여 리용되며 Explorer형식의 대면부를 제공한다.

엄격히 모든 URL들은 문법적으로 포구번호를 가져야 한다. 이전에 URL은 다음과 같은 형식으로 표현되였다.

```
http://java.sun.com:80/docs/book/tutorial/
```

보통 Web에서는 기정으로서 포구번호가 80으로 되여 있기때문에 URL문자렬에서 생략될수 있다.

그러나 보다 더 다양한 봉사를 위하여서는 서로 다른 포구번호를 리용하여야 할것이다. 즉 이 경우에는 항상 포구번호를 지적하여야 한다.

그러면 URL에서 http://라는 문자렬을 항상 기입하여야 하는가? 싸이트에 대한 FQDN은 홈페지를 보려고 하는 경우 http://문자렬을 요구할것이다. 만일 열람기로서 Netscape를 리용한다면 URL문자렬을 더 생략할수 있다. 즉 FQDN이 앞붙이 www와 뒤붙이 com을 가지고 있으면 이러한것들을 다 생략할수 있다. 생략형식으로서 간단히 redhat를 입력하면 확장된 형식 즉 http://www.redhat.com.으로 자동적으로 확장된다.

참고

~docs/index.html경로이름에서 문자 ~은 항상 삽입해 주어야 한다. UNIX체계는 ~문자를 홈등록부(17.8.3)로서 인식한다. 즉 실례에서 ~docs는 UNIX체계에서 /home/docs로 인식된다. 이 문자에 의하여 봉사기의 조작체계가 UNIX라는것을 알수 있다.

주의

URL문자렬은 대소문자를 완전히 구별하지 않는것은 아니다. 11.1.3에서 론의한바와 같이 FQDN은 대소문자를 구별하지 않는다. 즉 JAVA.SUN.COM과 java.sun.com은 서로 같은것이다. 그러나 파일경로이름은 대소문자를 구별할수도 있으며 조작체계에 의존할수도 있다. 대부분의 Web봉사기들은 UNIX를 리용하며 UNIX는 대소문자를 구별한다. 만일 경로이름이 /Docs/index.html이라면 그것을 그대로 입력하여야 한다.

14.11 Web의 언어 HTML

HTML이라는 략어를 풀어서 쓴다면 Hypertext Markup Language이다. 이것은 실지로 C언어나 Java언어와 같은 프로그람작성언어가 아니다. C언어와 Java언어는 본문원천으로부터 2진실행파일을 만든다. 사용자는 자기의 콤퓨터에 Web상의 HTML문서의 원천을 복사하여 볼수 있다. 이러한것으로 하여 이 언어는 배우기 더 쉽다. 추가적으로 HTML은 아래와 같은 특징들을 가지고 있으며 이것은 Web페지를 작성하는데 아주 적합하다.

- 가동환경에 무관계하다는것이다. 이 특징은 Windows, UNIX 혹은 Machintosh체계와 같은 그 어떤 체계에서도 페지의 내용을 볼수 있다는것이다.
- 열람기에 독립이라는것이다. 즉 어떤 문서를 Netscape나 Internet Explorer상의 어디에서 보든 비슷하다.
- 편집하기 대단히 쉽다는것이다. 즉 HTML이 본문파일로 되여 있기때문에 편집기를 리용하여 간단히 만들고 수정할수 있다. 그림이나 다매체물들은 서로 다른 파일로서 보존되여 있으며 문서파일은 그것들에 대한 련결로서 제공된다.
- 가장 최소화된 파일을 리용한다는것이다. HTML문서는 Microsoft Word의 .doc파일, Postscript의 .ps파일, Corel WordPerfect의 .wp와 같은 단어처리기의 크기에 대한 분모이다. 이 문서는 인터네트에서의 전송에 아주 적합하다. Web는 .html(그리고 .gif와 jpeg)파일이 크기로 볼 때 대단히 작기때문에 쉽게 관리한다.

HTML문서들은 확장자로서 .html을 리용한다. 그러나 대부분의 열람기들은 유효한 확장자로서 .htm을 리용한다. 이 언어는 실지 지령들을 리용하지 않고 그것이 매몰된 본문형식의 꼬리표를 리용한다. 지령 그자체는 보이지 않지만 이러한 꼬리표들은 본문의 형식과 구조를 처리한다. 이러한 문서는 2개의 서로 다른 체계에서 볼 때 꼭 같지는 않지만 서로 비슷하다. 왜 그렇게 되는가 하는것은 다음절에서 보기로 하자.

14.11.1 표식달기와 꼬리표

HTML의 시원은 1970년대, 80년대에 UNIX체계들에 표준장비되였던 nroff/troff지령에서부터였다. HTML은 본문을 형식화하기 위한 꼬리표(tag)들을 리용한다. 이 꼬리표들은 항상 기호 <로 시작하여 >로 끝나며 그의 대부분은 양식화기능을 가지고 있다. 실례로서 꼬리표 <I>와 이에 대응한 꼬리표 <\I>은 본문을 경사체로 표시하는 꼬리표이다. Protocol이라는 단어를 경사체로 표시하려면 두 꼬리표사이에 이 단어를 놓아야 한다. 즉 <I> protocol <\I>

모든 HTML문서는 본문에 표식을 붙이기 위한(이로부터 **표식달기**(markup)라는 용어가 나왔다.) 몇개의 꼬리표들로 이루어 져 있다. 사용자는 정확한 양식화속성을 지적하지 않으며 꼬리표로서 열람기에 알려 준다. 실례로 꼬리표 <H2>와 <\H2>를 리용하여 본문을 표식하면 이것은 굵은체로서 크기가 14인 Arial폰트로서 본문을 표시한다. 모든 열람기에서 내부해석은 일치하지 않다. Internet Explorer에서 페지는 Netscape에서의 페지와 서로 다르다.

주해

Netscape와 Microsoft는 자기자체의 꼬리표들을 표준적으로 가지고 있다. HTML 4.0은 표준적으로 모든 열람기에 대한 일부 공통적인 꼬리표들을 가지고 있으며 그외의 다른 꼬리표들을 허용하지 않는다.

14.11.2 HTML문서의 해석

HTML에 대한 기초지식을 주기 위하여 먼저 여기서 HTML문서가 어떤 형식으로 이루어 졌는가를 보자. 모든 html문서는 <HTML>꼬리표로부터 시작하여 <\HTML>꼬리표로 끝난다. 그다음 문서는 2개의 부분 즉 머리부와 본체로서 갈라 진다. 그림 14-8은 HTML문서를 보여 주며 그림 14-9와 같은 형식으로 화면에 나타난다.

```
<HTML>
  <HEAD>
       <TITLE>
           Perl: Larry Wall's Brainchild
       </TITLE>
  </HEAD>
  <BODY>
     <H1> Perl: Larry Wall's Brainchild </H1>
       <B>perl</B> is an interpretive language and is probably the
       best     language yet         available for text manipulation.
       <IMG SRC="pearl.gif" ALIGN=LEFT VSPACE=10 HSPACE=10>
       It was created by Larry Wall, and made freely available to the world.
       <EM><STRONG> You don't have to pay for using perl</STRONG></EM>,
       It's     distributed     under the GNU General Public License,
       which means that no one can impose any restrictions on its distribution.

       You can know more about <STRONG>perl</STRONG> by visiting
       <A HREF="http://www.perl.org"> the Perl site</A>
  </BODY>
</HTML>
```

그림 14-8. HTML문서의 본문

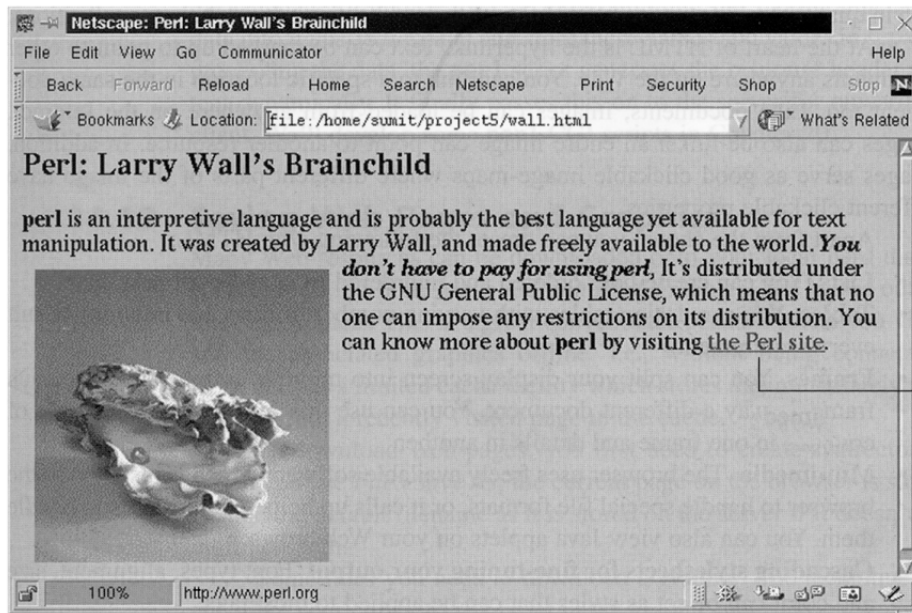



그림 14-9. Netscape에서 본 HTML문서

<HEAD>는 문서전체를 표현하는 추상화된 정보를 가지고 있다. 일부 꼬리표들은 <HEAD>꼬리표에 포함되여 있으며 대표적으로 <Title>꼬리표를 들수 있다. <TITLE>과 <\TITLE>사이에 삽입된 본문은 열람기의 제목띠에 나타난다. <HEAD> 꼬리표에는 또한 <BODY>와 <\BODY>꼬리표로 결정되는 본체가 포함된다. 여기에는 HTML의 모든 내용과 다른 모든 꼬리표들이 들어 있다.

열람기는 불필요한 공백들과 공백행들을 무시하며 또한 HTML문서안에 있는 공백은 그 수에 관계없이 하나의 공백으로 만들어 준다. <IMG>꼬리표는 페지에 그림을 배치하기 위하여 리용된다. 실례에서 전체 본문은 단하나의 단락으로서 나타나며 VSPACE와 HSPACE에서 지적된 크기를 가진 화상을 둘러 싼다.

HTML기능은 우에서 보여 주는것처럼 대단히 명백하다. 즉 본문과 그림들의 배치를 순전히 문서편집기로써 하였다.

우의 실례에서 the Perl site라는 단어는 특별히 아래에 밑선이 그어 져 있으며 또 다른 색으로 표시된다. 이것은 URL http://www.perl.org을 가리키는 하이퍼본문련결이다. 이러한 부분에 마우스를 가져 가면 유표가 손그림기호로서 변한다. 이때 아래의 상태띠에는 그에 해당한 URL이 나타난다. 만일 여기를 찰칵하면 www.perl.org의 홈페지가 열람기에 표시된다.

일부 꼬리표들은 자기의 속성을 가지고 있으며 속성값을 설정하는 일반적인 형식은 <속성=값>이다. 실례에서 SRC와 ALIGN은 <IMG>꼬리표의 속성들이며 이 속성들은 각각 pearl.gif와 LEFT로서 설정되여 있다. 꼬리표들과 그 속성들은 대소문자를 구별하지 않는다. 즉 <TITLE>과 <Tiltle> 그리고 <tiTle>은 열람기에 의하여 같은 방식으로 해석된다. 파일이름의 대소문자구별은 이와 같은 페지들을 관리하는 조작체계에 의존한다. UNIX는 대소문자를 구별하며 실례로 pearl.gif와 Pearl.gif는 서로 다른 파일로서 취급된다.

14.11.3 HTML의 능력

HTML은 꼬리표들로서 본문의 형식을 지적하는 표식달기언어이다. HTML은 일부 기초적인 단어처리능력을 가지고 있다. 사용자는 단락들로써 본문을 분할할수 있으며 또 굵은체와 경사체와 같은 시각적인 속성들을 거기에 추가할수 있다. 사용자는 6개의 서로 다른 크기를 가진 제목들을 제공할수 있다.

또한 GIF 혹은 JPEG형식의 그림들을 문서의 임의의 위치에 놓을수 있다. GIF와 JPEG는 크기를 최소로 하기 위하여 압축기술을 리용한다. GIF는 화상들은 256색으로서 표현하며 선을 그리거나 몇개의 색으로서 표현되는 작은 그림이나 아이콘들을 표시할 때 적용된다. 또한 JPEG화상은 1670만의 색으로서 표현되며 정밀한 화상을 표시할 때 리용된다. 그러나 JPEG의 압축기술은 손실이 많기때문에 화상이 표시될 때 정보루실로 하여 정확히 표시되지 않는 경우도 있다. 다른 형식의 IMAG들은 방조응용프로그람에 의하여 표시된다.

HTML에서 기본은 하이퍼련결이다. HTML문서에는 Web의 어떤 다른 문서를 지적하기 위한 본문이 포함되여 있다. 또한 같은 문서에서도 서로 다른 위치를 지적할수 있도록 련결시킬수 있다. 문서와 마찬가지로 화상들도 인터네트의 임의의 위치로부터 적재될수 있으며 그러한 매개의 화상들은 또 다른 자원과 련결될수 있다. 추가적으로 말한다면 이러한 화상들은 누르기가능한 화상배렬표(clickable image-map)로서 봉사한다. 사용자는 HTML을 리용하여 다음의것을 만들수 있다.

- **목록**(list): 사용자는 순서화된 목록과 비순서화된 목록을 만들수 있다.
- **표**(table): 행과 렬의 수로서 표를 설계할수 있다.
- **틀**(frame): 사용자는 여러개의 틀로서 화면을 분할할수 있다. 이 기능을 리용하여 여러개의 틀에 하나의 내용에 대하여 다른 형식으로서 표시할수 있다.
- **다매체**(multimedia): 열람기는 특별한 파일형식들을 처리하기 위하여 설치된 쏘프트웨어를 리용한다. 대표적으로 방조응용프로그람이다. 즉 Web열람기상에서 Java화상배렬표들을 볼수 있다.
- **출력조절을 위한 종속적인 양식표**: 폰트의 형, 줄맞추기, 크기, 색은 대부분의 꼬리표들에 적용할수 있는 형태이다.
- **매몰된 Java스크립트프로그람**: 사용자는 HTML문서에 Java스크립트프로그람을 매몰할수 있다. Java스크립트는 아주 편리한 프로그람작성구조를 가지고 있으며 양식자료를 검사하는데서 대단히 유용하다.

HTML은 확장성이 있다. HTML 4.0은 많은 꼬리표들을 추가할수 있게 되여 있다. 이것은 XML(Extended Markup Language)로서 언어 그자체를 갱신할수 있다는것을 말해 준다.

14.11.4 양식과 공통관문대면부

<FORM>꼬리표는 HTML과 Web에서 새로운 차원을 추가한다. 양식(form)들은 문서와의 대화를 제공하며 이것은 전자상업을 진행하기 위한 수단이다. 이 꼬리표는 검사칸, 단일선택단추, 내리펼침목록과 같은 보통의 Windows입력장치들을 리용할수 있게 한다. 사용자가 제공한 입력은 변수의 형태로 봉사기에 넘겨 진다. 사용자는 어떤 문서를 요구하거나 이 입력장치들을 통하여 제공된 어떤 자료를 처리하기 위하여 봉사기에 지시를 줄수 있다.

그러나 Web봉사기에는 이러한 처리를 진행하는 기능이 없다. 즉 봉사기는 자료처리요구를 받으면 일감을 처리하는 관문프로그람에 의존해야 한다. 이러한 봉사기측의 프로그람은 입력자료를 처리하고 HTML꼬리표들과 HTTP머리부들을 삽입하여 의뢰기측에서 볼수 있도록 동적으로 만들어 진 HTML내용을 반대로 보낸다. 이러한 기능을 수행하는 대표적인 프로그람은 **공통관문대면부**(Common Gateway Interface:**CGI**)프로그람이다. CGI프로그람을 실행하기 위한 명령은 <Form>꼬리표 그자체에 매몰된다.

```
<FORM METHOD="POST" ACTION="/cgi-bin/getname.pl">
```

이 경우에 봉사기는 getname.pl스크립트를 실행시켜야 한다. getname.pl은 perl(CGI프로그람작성

을 위한 언어)로써 씌여 진 하나의 프로그람이다. perl프로그람들은 이름-값형태로 쉽게 분리할수 있으며 서로 다른 변수에 이러한 이름들과 값들을 보존한다. 자료를 처리한후 perl스크립트는 응답머리부와 HTML내용을 동적으로 생성한다. 마지막으로 이 자료를 의뢰기에 넘겨 준다. 일부 perl의 CGI스크립트들은 제20장에서 더 심화시킨다.

14.12 Web페지와 도형의 보관

사용자는 많은 Web자원들을 자기의 하드디스크에 보관할수 있다. Web자원들은 HTML파일들과 정적 및 동적도형들을 포함하고 있다. 이러한것들은 Java애플레트들과 같이 실행될수 없는 자료이다. Web자원들을 하드디스크에 보관하면 인터네트에 접속하지 않고도 비직결방식으로 문서와 그와 련결된 도형들을 볼수 있는것으로 하여 대단히 편리하다. Netscape는 사용자가 본 싸이트에 대하여 즉시적으로 보존하는 완충능력을 가지고 있다. 그러나 사용자는 완충기에서 최근에 보았던 페지를 찾을수 없다.

Web페지들을 전송 받기 위하여서는 먼저 그것들을 보존하기 위한 등록부를 만들어야 한다. File>Save As를 선택하면 열람기에서 현재페지는 디스크에 보존된다. 사용자는 이때 기정파일이름을 그것이 의미를 나타내지 못한다면 자기 편리에 맞게 변경시킬수 있다.

때때로 사용자들은 도형을 전송 받아야 할 경우가 제기된다. 여기서 사용하는 기술은 여러가지이다. 이 경우에 화상의 어떤 위치에 마우스유표를 가져간 다음 오른쪽단추를 누르시오. 여기서 Save Image As를 선태하고 기정파일이름을 변경시키시오. 이렇게 하면 화상은 디스크에 보관된다. 만일 어떤 페지가 20개의 그림들을 가지고 있다면 우와 같은 조작을 20번 반복하여야 한다.

문제는 그러한 도형들을 식별하기 어렵다는것이다. 대단히 간단한 표라든가 푸른 배경색판의 본문블로크와 같은것들이 하나의 도형일수도 있다. 또한 작은 그림화상들에 대해서는 사람들이 대체로 주의를 돌리지 않는다. Netscape는 이와 같은 문제들에 대한 해결책을 가지고 있으며 여기서 간단히 취급한다.

때때로 페지에 동화상이 나타나는 경우가 있다. 이것은 GIF형식의 화상일수도 있으며 열람기에 내장된 Java가상기계에 의하여 실행되는 Java화상배렬표일수도 있다. 동화상들을 Shockwave로서 제작할수 있으며 이러한것들은 특별한 쏘프트웨어(삽입프로그람)에 의하여 볼수 있다. 이 경우 삽입프로그람(plugin)은 열람기에 설치되여 있어야 한다. 사용자는 여기서 서술된 기술을 리용하여 GIF형식의 동화상들은 보존할수 있지만 Java애플레트와 Shockwave파일들은 보존할수 없다.

참고

단순한 도형들을 식별하기 위한 가장 좋은 방법은 그러한 화상에 대하여 오른쪽단추를 누르는것이다. 이때 Save Image As항목이 나타나면 이것은 GIF 혹은 JPEG형식의 도형이라는것을 보여 준다.

14.12.1 내리적재된 파일의 보기

사용자는 인터네트와의 접속을 해제한 다음 File>Open을 리용하여 전송된 파일을 볼수 있다. 만일 그것이 HTML파일이라면 분명히 망에 접속하였을 때 보았던 문서일것이다. 사용자가 도형파일을 전송받았다면 이러한 파일을 볼수 있는가 하는것은 문서에서 <IMG>꼬리표가 사용되는 방법에 의존한다.

이것은 일정한 합성이 필요하다. 만일 <IMG>꼬리표가 http://www.sasol.com/pic s/wall.gif와 같은 절대경로를 가지고 있다면 비록 디스크에 이 파일이 전송되여 왔다 할지라도 열람기는 다시 Web로부터 그러한 경로의 파일을 불러 들이려 할것이다. 이것은 상대경로가 리용되지 않는 한 실천적이지 못하다.

IMG SRC="Wall.gif"(상대경로)와 같은 <IMG>참조는 도형이 HTML페지와 함께 같은 등록부에

그리고 같은 봉사기에 보존되여 있는 경우를 의미한다. 이 경우 열람기는 자동적으로 도형을 표시할수 있어야 한다.

국부파일은 알불이 file://규약을 리용하여 볼수 있다. File>Open은 사실상 이러한 규약을 리용하고 있다. 또한 사용자는 열람기로서 HTML파일을 보기 위하여 URL창문에 file://localhost/home/enry/download/wall.html와 같이 입력할수 있다. 만일 파일이 GIF 혹은 JPEG형식의 화상이면 이러한것들은 열람기에 나타난다. 사실상 이것은 도형파일을 보기 위한 가장 간단한 방법이다. 사용자에게는 실지로 열람기가 기동하고 있는 상태에서는 외부적인 프로그람은 필요없다.

만일 앞붙이 file://을 리용한 URL의 마지막요소가 등록부인 경우에는 어떻게 되겠는가? 이 경우에는 앞붙이 ftp://가 열람기창문(그림 11-2)에 파일들을 보여 주는 방법과 마찬가지로(그림 11-2) 목록화한 등록부를 보여 줄것이다. 즉 사용자는 Windows Explorer와 같은 대면부를 볼것이다. 여기서 등록부를 찰칵하면 그 등록부의 목록을 보여 줄것이다. 이것은 File>Open을 리용하는것보다 더 좋다.

참고

우에서 말한 앞붙이 file://을 리용한 URL은 localhost라는 이름을 FQDN으로 리용한다. 이 이름은 자기의 콤퓨터를 주소화하기 위하여 리용되는 이름이다(11.1.2에서 참고). localhost이름은 기호렬 ///을 리용하여 쓸수 있다. 즉 우에서 리용한 URL을 다시 쓰면 다음과 같다.

```
file:///home/henry/download/wall.html
```

3개의 사선기호대신에 한개의 사선기호를 리용하여 보시오. 그것이 동작하는가?

14.12.2 내리적재된 도형이 보이지 않을 때

때때로 사용자는 <IMG>꼬리표의 값으로서 pics/banner1.gif 혹은 ../../images/ anner1.gif와 같은 상대적인 경로지정을 볼수 있다. 이러한 파일참조들이 같은 문서에서 발견되는 경우 그것들은 같은 봉사기에 존재한다. 첫번째 경로에 해당한 파일은 HTML페지를 관리하는 등록부아래의 pics등록부에 있으며 두번째 경로에 해당한 파일은 문서등록부로부터 두 준위만큼 올라 가서 다시 image등록부로 한 준위만큼 내려 간 위치에 있다.

만일 GIF파일과 그와 련관된 HTML문서가 같은 등록부에 전송되여 왔다 하여도 도형파일들은 HTML문서를 볼 때 현시되지 않는다. 이러한것들을 보기 위하여서는 자기의 국부하드디스크에 같은 등록부구조를 가져야 한다. 특별히 이와 같은 등록부가 많다면 이것은 대단히 시끄러운 조작이다. 이 경우에는 HTML파일을 편집하여 속성값들에서 사선기호와 등록부이름들을 제거하는 편이 더 쉽다. 이러한 수정은 sed지령과 perl지령을 리용하면 간단히 할수 있다. 15.12.2에서 이와 같은 실례를 볼것이다.

14.12.3 Web페지를 도형과 함께 보관하기

하나의 조작으로 모든 도형들을 포함한 완전한 Web페지를 전송 받자면 어떻게 해야 하는가? 이것은 Netscape의 강력한 기능의 하나이다. 그러나 이 일감은 Navigator에 의해서가 아니라 Netscape Composer에 의하여 처리된다.

도형페지를 현시한 상태에서 File>Edit Page로서 Composer의 편집방식을 호출하시오. 이때 새로운 형의 창문이 나타나며 이 창문에 점차적으로 그림들이 표시된다. 이러한 그림들이 다 나타날 때까지 기다리지 말고 즉시 File>Save As를 선택하시오. 이때 기정적인 HTML파일이름이 등록부안의 이미 존재하는 파일이름들과 같지 않다면 이 파일이름을 그대로 리용하시오. Composer는 HTML파일뿐아니라 그와 련관된 도형자료 역시 보관한다.

14.13 열람기의 성능제고

폐지를 보는 방식은 사람들마다 다르다. 즉 어떤 사람들은 그림을 표시하지 않고 폐지를 보려고 하며 또한 Java애플레트들이 없이 보는것을 더 좋아 한다. 또 어떤 사람들은 대리자(proxy)봉사기를 리용하여 망의 여기저기를 탐색하려고 할것이다. 이 절에서는 열람기가 이러한 요구들에 어떻게 적응되는가 하는것을 보여 준다.

14.13.1 그림과 애플레트가 필요 없을 때

그림이나 Java애플레트들을 적재하는것은 시간이 걸린다. 이런 경우에 사용자는 열람기가 이와 같은 매체자료를 호출하지 않게 할수 있다. 즉 Edit>Preferences >Advanced를 선택한 다음 Automatically load와 Enable Java칸의 검사표식을 해제하시오. 즉 Java스크립트를 보이지 않게 하는것으로써 시간을 더욱 단축할수 있다.

열람기는 이때부터 화면에 본문만을 표시하며 화상들은 표시하지 않는다. 반대로 폐지의 모든 화상들을 볼 필요성이 제기되면 Navigation도구띠에서 Images단추를 눌러야 한다. 또한 표시되여야 할 위치에 대하여 마우스단추를 누르면 그 화상이 회복된다. 즉 이 위치에 대하여 마우스를 한번 누르면 화상을 표시하며 다시한번 누르면 그와 련결된 자원을 불러 들인다.

14.13.2 완충기의 리용

매개 열람기들은 현재의 Web자원을 보관하기 위한 주기억완충기와 디스크완충기를 가지고 있다. 사용자가 URL을 입력하면 열람기는 먼저 이 완충기에 적재한다. 디스크완충기는 앞으로의 대화에 리용될수 있지만 기억기완충기는 명백히 그렇게 할수 없다. 어떤 큰 완충기는 많은 폐지들을 보관할수 있다. 그렇지만 이러한 폐지들을 회복하는것은 시간이 걸린다. 사용자가 32MB의 RAM기억기를 가진 콤퓨터를 사용한다면 기억기완충기와 디스크완충기는 각각 4M, 20M이상 리용할수 없다. 사용자는 Edit>Preference>Advanced>Cache를 사용하여 Netscape의 완충기크기를 재정의 할수 있다.

대부분의 Web페지들은 정적인 내용을 가지고 있다. 만일 사용자가 이미 보존된 완충기의 내용을 직접 보려고 한다면 이 내용들은 회복되지 않는다. 이때에는 완충기안의 문서를 회복하도록 하는 단일선택단추를 선택하는것이 필요하다.

참고

사용자는 때때로 대리자 혹은 Netscape의 내부완충기에서가 아니라 망으로부터 다시 자료를 불러 들일수 있다. 즉 완충기로부터 불러들일수 없는 경우에 Reload단추를 누른 상태에서 shift건을 누르시오.

14.13.3 대리자를 통한 접근

망에 가입한 상태에서 사용자는 관문주콤퓨터(gateway host)를 통해서만이 인터네트에 접근할수 있다. 이러한 콤퓨터는 **대리자**(proxy)라는 쏘프트웨어를 실행하고 있다. 즉 사용자는 이러한 대리자를 통하여 인터네트에 접근해야 한다. 대리자봉사기는 사용자가 요구하는 폐지를 망으로부터 얻어 내며 이를 위하여 큰 완충기를 가지고 있다. 그러므로 완충기에 이미 적재된 폐지에 한해서는 그 호출이 대단히 빠르다. 이러한 폐지들은 여러날동안 완충기에 남아 있게 된다.

관문콤퓨터의 IP주소는 192.168.0.100이며 Squid대리자봉사기를 사용하고 있다. 사용자는 포구번호

3128(Squid에 의하여 리용되는 기정포구번호)로서 대리자를 공유하여 통신하여야 한다. Edit>Preferences>Advanced>Proxies를 선택하시오. 그다음 여기서 Manual proxy configuration을 선택하고 View를 누르시오. 그리고 HTTP Proxy:에 192. 168.0.100 혹은 FQDN을 포구번호로서는 3128을 입력하시오. 이렇게 함으로써 사용자는 대리자봉사를 리용할수 있는 준비를 갖추게 된다.

14.14 터지지 않는 풍선

스코트 맥닐리(Scott McNealy)는 한때 《망은 콤퓨터이다.》라고 말하였는데 그때는 극소수의 사람들이 그가 말한 의미를 리해하였다. 세계를 더욱 친밀하게 해준 협력정신을 촉진시키는것으로 하여 망은 더욱더 장성하였다. 이러한 협력정신에 의하여 이미 Linux라는 이름을 가진 우수한 제품이 생산되였다. 그리고 수십억을 넘는 정적인 Web페지들이 만들어 졌다. 모든 주제에 관한 정보들이 매일 추가되고 있으며 수많은 질 좋은 프리웨어(freeware), 쉐어웨어(shareware)프로그람들이 쓸모 있는것으로 되고 있다. 앞으로 더많이 생겨 날것이다.

크리스토퍼 앤더슨은 인터네트가 《정보초고속도로의 거대한 모형을 시도하였다.》고 하였다. 회사들이 Web에 개입함에 따라 사람들은 개방형규격에서 작업하던 설계가들의 핵심그룹이 혹시 주주들의 공격에 부닥치게 되지 않겠는가고 생각하고 있다. 그러나 우리가 보다 자신심 있게 개방형규격, 비독점적인 규격을 유지한다면 인터네트는 그야말로 《터지지 않는 풍선》이 될것이다.

14.15 Web상에서의 MIME기술

Web페지의 구성요소로서 도형을 도입하였을 때 그에 대한 선택권은 완전히 제한되여 있었다. 그때부터 모든 열람기들이 GIF와 JPEG파일들을 외부적인 프로그람을 리용하지 않고 직접 표시하는것을 기대하게 되였다. Web에 대한 기대는 더욱 늘어 났으며 그로 하여 여러가지 종류의 매체를 가지게 되였다. 현재에는 Java, RealAudio, RealVideo, Shockwave기술이 존재한다. 이 기술로 동화상 ,음성, 비데오를 자유롭게 볼수 있다.

비록 열람기가 많은 기능들을 내장하고 있다 해도 이러한 매체들을 다 관리할수는 없다. 그러므로 열람기는 자기의 삽입프로그람(plugin) 혹은 방조응용프로그람(helper application)을 호출해야 한다. 방조응용프로그람은 외부응용프로그람이며 이 프로그람을 적재하는것은 오랜 시간이 걸린다. 더우기 편집(editing), 구성(composing)기능을 가진 전체 응용프로그람을 호출하는것은 음성파일을 듣는다거나 어떤 파일을 보려고 하는 경우 지장을 줄것이다. Web는 삽입프로그람에 의하여 이러한 문제를 쉽게 해결한다.

14.15.1 삽입프로그람

삽입프로그람(plugin)은 열람기에 적재되는 쏘프트웨어의 한 부분이다. 이것은 크기가 보통 작으며 파일을 보기(음성, 비데오의 재생) 위한 최소한의 기능을 가지고 있다. 사용자는 WordPerFect와 같이 방조응용프로그람을 호출하는것과는 다르게 개별적인 삽입프로그람을 호출할수 없다. 삽입프로그람으로서 어떤 파일을 보는 경우 이것은 다른 창문에서가 아니라 HTML본문과 함께 직접 나타난다.

Linux콤퓨터에 Shockwave삽입프로그람을 적재하기 위해서는 Web(http://www. shockwe.com)로부터 삽입쏘프트웨어를 불러 들여야 한다. Tar지령(22.9)을 리용하여 그 내용들을 불러 들인 다음 .netscape/plugins등록부에 아래와 같은 2개의 파일을 간단히 복사하시오.

```
cp ShockwaveFlash.class $HOME/.netscape/plugins
cp libflashplayer.so $HOME/.netscape/plugins
```

$HOME/.netscape/plugins등록부에 적재된 삽입프로그람들은 Netscape와 함께 등록된다. 이러한 삽입프로그람들을 보기 위하여서는 간단히 URL창문에 다음과 같이 입력하시오.

```
about:plugins[Enter]
```

Netscape는 또한 열람기상에서 Java화상배렬표들을 볼수 있게 하는 Java삽입프로그람과 함께 제공된다. Java와 Shockwave Flash외에도 많은 삽입프로그람들이 제작되고 있다.

14.15.2 방조응용프로그람

열람기는 자기가 관리할수 없는 자료형식을 만나면 먼저 그에 해당한 삽입프로그람이 있는가를 검사한다. 만일 없다면 열람기의 구성방식에 의존하며 Netscape는 아래와 같은 두개의 선택권한을 준다.

- **방조응용프로그람**을 지적한 다음 파일을 여시오.
- 후에 그 파일을 리용할수 있도록 디스크에 보존하시오.

전자우편에서 리용된 MIME기술(13.9)은 Web의 다매체파일에 대하여서도 적용된다. 그러나 이러한 파일들은 여러개의 통보문으로서가 아니라 독립적인 파일로서 Web봉사기에 의하여 보내진다. 봉사기는 파일을 보내기전에 먼저 자료기지를 탐색하여 그 파일의 확장자에 해당한 내용형을 의뢰기에 보낸다. 의뢰기는 이러한 내용형에 해당한 정확한 방조응용프로그람을 찾기 위하여 또 다른 자료기지를 탐색한다. 이러한 자료기지들은 실제로 간단한 구조를 가지는 2개의 본문파일이다.

- /etc/mime.types 혹은 $HOME/.mime.types(Netscape에 의하여 리용됨)
- /etc/mailcap 혹은 $HOME/.mailcap(Netscape에 의하여 리용됨)

우의 첫번째 파일은 Web봉사기에 의하여 사용되며 때때로 의뢰기에 의해서도 리용된다. 이 파일은 모든 내용형에 파일확장자들을 반영하고 있다. 아래에 보여 주는것은 RealPlayer에 대한 3개의 가능한 내용형들이다.

```
audio/x-pn-realaudio ra rm ram
audio/vnd.rn-realaudio ra rm ram
application/smil smi
```

우의 내용형들은 서로 다른 확장자를 반영하고 있다. 실례로 봉사기는 .ra확장자에 대하여 먼저 파일 mime.types을 참조하며 우의 첫행에서 보여 주는 내용형과 음성파일을 련결시킨다. 다음 봉사기는 파일을 보내기전에 의뢰기에 내용형머리부를 보낸다.

주해

열람기가 ftp를 리용하여(ftp://) 혹은 국부적으로(file://) 파일을 불러 들이는 경우에는 파일내용에 앞서 내용형이라는것이 존재하지 않는다. 이 경우에 열람기는 확장자를 보고 파일의 형을 추측할수 있어야 하며 이를 위하여 mime.types파일을 요구한다. 이러한 규약을 리용하면 사용자는 열람기창문에서 파일아이콘을 눌러 그에 해당한 내용을 볼수 있다.

의뢰기측에서 열람기는 방조응용프로그람을 찾기 위하여 파일 mailcap에서 audio/x-pn-realaudio가 있는가를 탐색한다. Mailcap의 RealPlayer자료에 대한 일치하는 항목들은 다음과 같다.

```
audio/x-pn-realaudio; /usr/local/RealPlay7/realplay %u
audio/vnd.rn-realaudio; /usr/local/RealPlay7/realplay %u
application/smil; /usr/local/RealPlay7/realply %u
```

두번째 마당은 방조응용프로그람으로서 realplay를 지적한다. 열람기는 파일을 재생하기 위하여 이 응용프로그람을 호출한다. 방조응용프로그람은 현재 MP3과 Shockwave Flash파일들을 포함하는 다매체 자료형들을 관리한다.

Netscape에서는 자료기지파일로서 홈등록부의 .mime.types와 .mailcap을 리용한다. 이러한 파일들은 차림표의 Edit>Preference>Navigator>Applications를 리용하여 열람기를 구성한다면 Netscape에 의하여 직접 편집된다.

Realplayer방조응용프로그람을 설치하기 위해서는 Red Hat Linux에 대한 rp7_ redhat6.bin이라는 실행프로그람을 불러 들여야 하며 그것을 뿌리사용자로서 실행시켜야 한다. 기정으로서 Realplayer방조응용프로그람은 /usr/local/RealPlayer7에 설치된다. 열람기구성의 상세한 내용을 보기 위하여서는 README파일을 참고하시오.

주해

파일 mime.types는 파일이름확장자들에 대한 목록을 포함하고 있다. 파일 mailcap는 지적된 확장자를 가진 파일을 보기 위하여 리용되는 helper응용프로그람의 이름을 포함하고 있다. 2개의 파일들은 다 Content-Type를 포함하고 있다.

요 약

인터네트는 TCP/IP통신규약을 리용하는 망들의 집합이라고 말할수 있다. ftp와 telnet외에도 추가적으로 인터네트는 우편목록, 새소식그룹, IRC, WWW를 제공한다.

인터네트의 이름공간은 령역들과 부분령역들로 나누어 진다. 웃준위령역들은 com, net, edu, org와 같은 3개의 문자로 이루어 진 일반령역이름들을 가진다. 모든 나라들은 2개의 문자로 이루어 진 웃준위령역이름을 가진다. 부분령역을 만들기 위한 권한은 매개 나라들에 국한된다.

우편목록은 목록의 모든 성원들에게 자동적으로 통보문을 보내기 위한 어떤 프로그람에 의하여 관리된다. 가입요구와 정보자료들은 통보문 그자체가 목록주소에 보내지는것과 동시에 관리자에게 보내진다.

새소식그룹들은 포구번호 119에서 NNTP통신규약을 리용하여 새소식을 봉사한다. 사용자는 새소식봉사를 받기 위하여 새소식그룹에 가입하여야 한다. 새소식그룹은 령역 이름과 같이 계층적인 구조로 되여 있는 이름을 가진다. 새소식읽기프로그람은 먼저 머리부파일들을 꺼내며 선택된 머리부들에 한해서만 그 기사를 불러 들인다. Tin은 성능이 강한 새소식읽기프로그람으로서 문자방식에 기초하고 있다. 그러나 새소식을 관리하기 위하여 Netscape를 리용할수도 있다.

인터네트중계담화(IRC)는 여러명의 사용자들이 통로들에 접속하여 담화를 진행할수 있게 한다. 이때 모든 사용자들은 자기의 유일한 별명을 가진다. 통로에 보내진 통보문은 /msg 혹은 /query지령이 리용되지 않는다면 그러한 통로의 모든 사용자들에게 가닿는다. 사용자는 또한 봉사기를 무시하고 /dcc지령으로써 직접 련결하여 파일을 교환할수 있다.

Web는 하나의 문서를 인터네트의 어떤 문서 혹은 자원과 련결하기 위하여 하이퍼본문이라는것을 리용한다. 어떤 문서는 그림들을 반영할수 있으며 또 그것들은 다른 문서 혹은 그림들과 련결시킬수 있다. 열람기는 이러한 련결로서 지적되는 모든 문서들과 그림들을 다 보여 준다.

문서는 경로이름과 싸이트의 FQDN을 결합한 URL을 리용하여 호출된다. URL들은 또한 서표가 붙

을수 있다. 이러한 서표화된 URL들에 대해서는 후에 더빨리 호출할수 있다. Netscape는 사용자가 본 모든 페지들에 대한 URL들을 보존한다. 또한 URL들은 ftp싸이트와 telnet싸이트를 지적할수 있다.

Web는 포구번호 80에서 하이퍼본문전송통신규약(HTTP)을 리용하여 작업하며 8bit자료를 전송할수 있다. HTTP는 이전 접속에 대한 지식이 없이도 하나의 접속으로서 자원을 호출할수 있는 무상태통신규약이다. 봉사기는 문서에 대한 내용형를 지적하는 응답머리부를 보낸다.

Web문서는 HTML언어를 리용하여 작성된다. HTML문서는 자동환경에 무관계하다. 이 언어로서 같은 문서안에서 본문과 그림의 위치를 지적할수 있다. GIF와 JPEG형식의 화상들은 열람기에 의하여 직접 표시된다.

HTTP는 양식과 함께 리용되는 경우 다르게 동작한다. 양식자료는 일반적으로 공통관문대면부(CGI)에 넘겨진다. 자료를 처리하는 이 프로그람은 HTML코드를 만들고 봉사기에 그것을 돌려 준다. perl은 CGI프로그람작성을 위한 언어이다.

모든 문서들과 도형들은 비직결방식에서 표시되도록 국부콤퓨터에 보관할수 있다. 전송된 파일은 file://형식의 URL을 리용하여 볼수 있다. 문서의 도형들은 <IMG>꼬리표에 상대경로로 지적되여도 화면에 표시된다. Netscape Composer를 리용하여 사용자는 경로이름에 대하여 생각하지 않아도 도형들을 포함한 전체 문서를 보존할수 있다.

열람기의 성능을 제고하는것으로써 사용자는 그림 혹은 Java를 보이지 않게 할수 있다. 사용자는 완충기의 크기를 조절할수 있으며 대리자봉사기를 리용하여 Web를 호출할수 있다. 또한 완충기로부터가 아니라 망으로부터 문서를 다시 불러 들이기 위하여 Reload단추와 함께 [Shift]건을 리용할수 있다.

열람기는 자기가 관리할수 없는 자료형식에 대해서는 삽입프로그람 혹은 방조응용프로그람을 호출한다. 방조응용프로그람과는 다르게 삽입프로그람은 열람기에서만 호출되는 쏘프트웨어의 한 부분이다. 봉사기는 파일확장자로부터 내용형을 결정하기 위하여 mime.types파일을 검색한다. 다음 열람기는 내용형에 대응한 방조응용프로그람을 찾기 위하여 mailcap파일을 탐색한다.

시험문제

1. 다른 나라들의 기관들에 대하여 리용되는 3개의 일반령역이름을 쓰시오.
2. 아래의 주소에 자기의 질문을 보낼수 있는가?

   `listserv@lists.colorado.edu`
3. 새소식통보문은 왜 리력에 보관되여야 하는가?
4. 어떤 새소식그룹에 통보문을 보낼수 있는가 없는가를 검사하기 위해서는 먼저 어디에 통보문을 보내야 하는가?
5. FAQ란 무엇인가?
6. IRC를 리용하여 2명의 사용자가 2대의 서로 다른 봉사기에 접속하여 통신을 유지할수 있는가?
7. lynx의 성능이 왜 Netscape의 성능보다 우월한가?
8. HTML문서를 호출하기 위해서는 어떤 Web통신규약과 포구번호를 리용해야 하는가?
9. Netscape Navigator를 리용하여 중간페지를 거치지 않고 이전에 자기가 본 페지를 어떻게 찾을수 있는가?
10. 어떤 Web싸이트를 주기적으로 보려고 한다면 그것을 더빨리 호출하기 위해서는 어떻게 해야 하는가?
11. 하이퍼본문은 본문만으로 이루어 졌는가?
12. URL로서 FQDN만을 리용한다면 열람기에 의하여 보게 되는 페지의 기정파일이름은 무엇인가?

13. 자기 콤퓨터에서 실행되는 Web봉사기의 홈페지를 보려면 어떻게 해야 하는가?
14. HTTP은 왜 무상태규약이라고 하는가?
15. Web페지들에 대한 내용형은 무엇인가?
16. HTML문서는 서로 다른 열람기에서 왜 다르게 보이는가?
17. 누르기가능한 화상배렬표란 무엇인가?
18. Netscape로 보게 되는 Web페지에서 도형을 어떻게 식별하는가?

련습문제

1. telnet를 리용하여 열람기로부터 싸이트 patvolkcal.com에 어떻게 접속하는가?
2. 우편목록이란 정확히 무엇이며 그것은 별명(alias)과 어떻게 다른가?
3. 새소식봉사가 주콤퓨터 news.planets.com에서 가능한가 하는것을 어떻게 빨리 검사할수 있겠는가?
4. 새소식그룹통보문은 전자우편통보문과 어떻게 다른가?
5. IRC로서 어떤 사용자와 통신하려고 한다. 그를 통로에 초청하기전에 비밀통보문을 보내기 위해서는 어떻게 해야 하는가?
6. IRC를 리용하여 어떤 사용자에게 파일 woaniak.gif를 직접 보내려고 한다. 어떻게 해야 하는가?
7. Netscape열람기는 외부적인 도움이 없이 임의의 형식의 그림을 표시할수 있는가? 그림들은 어떤 창문에 나타나는가?
8. 하이퍼본문이란 무엇인가?
9. URL에서 단어 Uniform의 의미는 무엇인가?
10. URL로서 www.planets.com/catalog.html대신에 WWW.PLANETS.COM/CAT ALOG.HTML을 리용할수 있는가?
11. FQDN www.planets.com을 가진 주콤퓨터안의 /etc등록부에 파일 foo.html을 놓는다면 열람기로부터 URL http://www.planets.com/etc/foo.html으로서 그 파일을 호출할수 있는가?
12. http://www.ge.com싸이트에 접속하기 위하여 입력할 URL의 생략된 최소문자렬은 무엇인가?
13. 10개의 도형을 가진 Web페지를 전송 받기 위하여 HTTP는 얼마나 많은 접속을 요구하는가?
14. 왜 HTML이 Web에 대하여 특별히 적합한가?
15. 열람기가 양식을 리용한 자료를 봉사기에 넘긴다면 봉사기는 그 자료를 어떻게 관리하는가?
16. 열람기에서 Reload단추는 어느 때 필요하며 그 기능이 제대로 수행되지 않는 경우 어떻게 해야 하는가?
17. 모든 도형들과 Web페지전체를 하나의 방향으로 보존하기 위해서는 어떻게 해야 하는가?
18. Web페지를 불러 들인 다음 페지의 모든 도형들을 개별적으로 보존하였다고 하자. 이 경우 도형들은 비직결방식에서는 나타나지 않는다. 원인은 무엇인가?
19. 어떤 위치에 대하여 마우스를 누르면 우편편집창문을 표시하려고 한다. 그러면 HTML문서에 어떤 명령문을 놓아야 하는가?
20. HTTP통신규약이 8bit파일들을 관리할수 있다면 왜 MIME기술이 필요한가?
21. 삽입프로그람이란 무엇인가? 또 방조응용프로그람과 무엇이 다른가?

제 15 장. 정규식을 리용한 려과기 grep와 sed

사용자들은 흔히 어떤 패턴에 따라 파일을 탐색하려고 한다. 실례로 어떤 패턴을 포함하고 있는 행들만을 표시한다거나 어떤 문자렬로서 지적된 패턴을 다른 문자렬로 교체하려고 한다. 이 장에서는 우와 같은 기능을 가진 2개의 중요한 려파기들로서 grep와 sed에 대하여 론의한다. grep는 사용자들에게서 제기되는 모든 탐색요구들을 관리하며 sed는 한행에서 개별적인 문자들을 관리할수도 있다. 다시 말하여 grep보다 더 구체적인 처리를 진행한다. 사실상 sed는 여러가지 처리능력을 가지고 있다.

이 장에서는 또한 UNIX의 중요한 특징의 하나로서 정규식에 대하여 취급한다. 이러한 표현식에 대해서는 vi와 emacs의 탐색기능을 리용할 때 간단히 보았다. 그러나 앞에서 보다 더 구체적으로 즉 가능한 모든 경우의 표현식들을 본다. 이 장에는 표현식작성규칙들이 정의되여 있으며 이 규칙을 리용하여 품을 들이지 않고도 패턴비교를 수행할수 있는 압축된 표현식들을 이끌어 낼수 있다. 사실상 grep와 sed코드의 한행은 C코드의 여러행에 대응된다.

체계관리자는 정규식들을 리해하고 작성하는데 습관되여야 한다. grep, sed와 함께 표현식의 사용법을 배우는것은 perl을 리해하는데서도 큰 도움을 줄것이다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- grep로 간단한 문자렬을 포함하고 있는 행들만을 표시한다(15.2).
- 행의 수, 행번호, 패턴을 포함하지 않는 행들을 표시한다(15.3).
- 정규식을 리용하여 서로 비슷한 문자렬을 탐색한다(15.4).
- 여러개의 패턴들을 리용하는 egrep와 fgrep의 리용방법을 배운다(15.5).
- egrep에서 리용하는 특수한 정규식에 대하여 배운다(15.6).
- sed를 리용하여 행을 선택하고 편집한다(15.7부터 15.10까지).
- 정규식을 리용하여 sed로써 하나의 패턴을 다른 패턴으로 바꾼다(15.11).
- 구간 및 꼬리표 붙은 정규식을 리용하여 grep와 sed의 기능을 강화한다(15.12).

15.1 실례자료기지

이 장에서와 려파기들, 쉘프로그람작성에 대하여 취급하는 다른 장들에서 자주 보게 되는 파일 emp.lst는 이 책에서 실례로 리용되는 자료기지파일이다. 또한 때때로 자료기지파일로서 이 파일로부터 유도된 또 다른 파일을 리용한다. 이 파일에서 매개 마당들은 명백히 구분되여 있으며 의미를 가지고 있다.

```
$ cat emp.lst
2233|charles harris   |g.m.     |sales      |12/12/52| 90000
9876|bill johnson     |director |production |03/12/50|130000
5678|robert dylan     |d.g.m    |marketing  |04/19/43| 85000
2365|john woodcock    |director |personnel  |05/11/47|120000
5423|barry wood       |chairman |admin      |08/30/56|160000
```

```
1006|gordon lightfoot  |director |sales      |09/03/38|140000
6213|michael lennon    |g.m      |accounts   |06/05/62|105000
1265|p.j. woodhouse    |manager  |sales      |09/12/63| 90000
4290|neil o'bryan      |executive|production |09/07/50| 65000
2476|jackie wodehouse  |manager  |sales      |05/01/59|110000
6521|derryk o'brien    |director |marketing  |09/26/45|125000
3212|bill wilcocks     |d.g.m.   |accounts   |12/12/55| 85000
3564|ronie trueman     |executive|personnel  |07/06/47|750000
2345|james wilcox      |g.m.     |marketing  |03/12/45|110000
0110|julie truman      |g.m.     |marketing  |12/31/40| 95000
```

이 파일에서 처음 5개의 행은 sort에 대하여 서술한 9.12에서 리용되는 파일 shortlist와 같다. 매개마당들에 대해서도 이미 설명되였다. 그러나 여기서 편리상 다시 설명하기로 한다. 이 파일은 종업원자료기지로서 15명에 해당한 내용을 보여 주는 본문파일이다. 파일은 마당구분문자 |를 리용하여 6개의 마당 즉 종업원식별번호, 이름, 성별, 직위, 난날, 로임마당으로 이루어 져 있다. 마당구분문자 |는 쉘에서 특별한 의미를 가진다. 그러므로 구분문자를 지적할 때 그것을 \으로서 의미해제해야 한다.

15.2 패턴에 의한 탐색(grep)

UNIX는 탐색요구를 조종하기 위한 특수한 계렬의 지령들을 가지고 있다. 여기서 grep지령은 이 계렬의 주요한것들중의 하나이다. 이 지령은 먼저 패턴비교를 위한 파일을 접수한 다음 리용된 선택항목에 따라 그 내용을 표시한다.

- 선택된 패턴을 포함하고 있는 행들을 표시한다.
- 선택된 패턴을 포함하고 있지 않는 행들을 표시한다(-v).
- 패턴이 나타나는 행번호들을 표시한다(-n).
- 패턴이 들어 있는 행의 수를 표시한다(-c).
- 패턴이 나타나는 파일이름들을 표시한다(-l).

grep의 리용방법 역시 대단히 간단하다. 문법적으로 첫번째 인수는 패턴에 해당되며 나머지인수들은 파일이름이다.

```
Grep options pattern filename(s)
```

정규식에 대해서는 후에 보기로 하고 먼저 탐색패턴으로서 간단한 문자렬을 리용하여 보자. 그러면 실례로서 emp.lst파일로부터 패턴 sales를 포함하고 있는 행들을 grep지령이 어떻게 표시하는가를 보자.

```
$ grep sales emp.lst
    2233|charles harris    |g.m.     |sales     |12/12/52| 90000
    1006|gordon lightfoot  |director |sales     |09/03/38|140000
    1265|p.j. woodhouse    |manager  |sales     |09/12/63| 90000
    2476|jackie wodehouse  |manager  |sales     |05/01/59|110000
```

여기서는 패턴에 인용부호를 사용하지 않았다. 인용부호는 탐색문자렬이 여러개의 단어들로 이루어졌거나 *,?와 같은 쉘문자들을 리용하는 경우에 사용한다. 인용부호를 리용하였다고 하여 처리가 달라

지는것은 아니다. 즉 grep "sales" emp.lst지령은 우와 같은 방식으로 동작한다.

grep는 하나의 려과기능을 수행하기때문에 패턴에 대하여 표준입력자료를 탐색하여 그 출력자료를 하나의 파일로서 보존할수 있다.

```
who | grep henry>foo
```

grep가 문자렬들의 계렬과 함께 리용될 때 첫번째 파라메터는 패턴으로, 나머지는 파일이름들로서 해석된다. 표시되는 모든 행들에는 그앞에 파일이름이 붙는다.

```
$ grep director empl.list emp2.lst
emp1.lst:1006|gordon lightfoot |director   |sales      |09/03/38|140000
emp1.lst:6521|derryk o'brien   |director   |marketing  |09/26/45|125000
emp2.lst:9876|bill johnson     |director   |production |03/12/50|130000
emp3.lst:2365|john woodcock    |director   |personnel  |05/11/47|120000
```

15.2.1 grep에서의 인용부호

패턴으로서 여러개의 문자렬을 리용한다면 거기에 인용부호를 붙여야 한다. 만일 그렇게 하지 않으면 첫번째 문자렬은 패턴으로, 나머지문자렬들은 파일이름으로서 취급된다. Gordon lightfoot을 패턴으로 하여 grep지령을 리용하면 다음과 같이 표시된다.

```
$ grep gordon lightfoot emp.lst
grep: lightfoot: No such file or directory
emp.lst:1006|gordon lightfoot  |director  |sales    |09/03/38|140000
```

grep는 lightfoot를 파일이름으로서 인식하며 분명히 이러한 파일을 여는것은 실패로 된다. 그러나 탐색은 다음인수로서 즉 emp.lst를 리용하는것에 의하여 계속된다. 이번에는 패턴에 인용부호를 붙여 보자.

```
$ grep gordon lightfoot emp.lst
1006|gordon lightfoot  |director  |sales     |09/03/38|140000
```

우의 결과는 정확하다. 또 이번에는 패턴으로서 neil o'bryan이라는 문자렬을 리용하여 보자.

```
$ grep 'neil o' bryan' emp.lst
>
```

Bourne, Korn 그리고 bash쉘들은 프롬프트문자 >을 내보내는것으로써 미완성된 지령으로 해석한다. 이 지령은 오유가 있는 경우에도 C쉘에서는 실행된다.

```
% grep 'neil o'bryan' emp.lst
Unmatched '.
```

우의 실례에서는 패턴 그자체에 외인용부호가 포함되여 있다. 쉘은 패턴의 범위를 결정하기 위하여 짝수개의 인용부호가 있는가를 검사한다. 실례에서는 겹인용부호가 3개 있으므로 여기서는 패턴이 정확히 구분되지 않는다. 이 경우에는 겹인용부호를 리용해야 한다.

```
$ grep "meil o'bryan" emp.lst
4290|neil o'bryan     |executive  |production |09/07/50|  65000
```

겹인용부호를 리용하면 패턴에 겹인용부호가 여러개 있다 하여도 오유로 되지 않는다. 반대로 패턴에 겹인용부호가 포함되여 있다면 이때에는 겹인용부호로서 둘러 막는다. 사용자는 이러한 인용부호안에 정규식을 놓을수도 있다.

주해

만일 패턴이 둘이상의 단어 혹은 쉘에 의하여 다르게 해석되는 특수한 문자들을 포함하고 있다면 패턴에 항상 인용부호를 붙여야 한다. 일반적으로 패턴에는 외인용부호 혹은 겹인용부호들을 리용할수 있지만 지령대입이나 변수값을 평가하는 경우에는 겹인용부호를 붙여야 한다.

15.2.2 grep가 실패하였을 때

grep는 UNIX의 대표적인 지령이며 패턴을 찾을수 없는 경우에 간단히 프롬프트를 돌려 준다.

```
$ grep president emp.lst
$_
```

지령은 문자렬 president를 찾을수 없기때문에 실패하였다. 이러한 패턴탐색기능은 sed와 awk지령에서도 리용되지만 이와 같은 지령들은 패턴을 찾을수 없는 경우에 실패로서 귀환하지 않는다.

그러나 우의 실례는 결코 틀린 결과를 나타내지 않는다. 즉 실패로 되는 근거는 없다. 사실상 이러한 cmp조작은 성공을 나타낸다. 성공인가 혹은 실패인가 하는것은 지령의 실행이 끝난후 설정되는 특수한 변수 $?의 값에 의하여 결정된다. 이 변수의 리용에 대하여서는 쉘을 취급하는 18.5.1에서 본다.

15.3 grep의 선택항목

grep지령은 자주 리용되는 대표적인 UNIX지령의 하나이다. 이 지령은 몇개의 선택항목들을 가지고 있으며(표 15-1) 사용자는 이 지령의 대부분의 선택항목들에 대하여 잘 알아야 한다. 선택항목들은 그리 많지 않다.

표 15-1. grep계렬에서 리용되는 선택항목들

선택항목	의 미
-c	출현수를 표시한다.
-l	파일이름만으로 된 목록을 표시한다.
-n	행번호와 함께 보여 준다.
-v	표현식에 맞는 행들을 보여 주지 않는다,
-I	비교할 때 대소문자를 무시한다.
-h	여러개의 파일을 관리하는 파일이름을 생략한다.
-w	완전한 단어를 비교한다.
-e pat	기호 -로 시작하는 패턴 pat를 비교한다.
-e pat	우와 같지만 여러번 사용될수 있다(Linux와 일부 UNIX판에서 리용).
-E	패턴을 egrep의 정규식으로서 취급한다.
-F	fgrep형태로서 패턴을 비교한다.
-n	비교된 행을 기준으로 우아래로 각각 n개 행씩 보여 준다.
-A n	비교된 행의 아래로 n개 행을 더 보여 준다.
-B n	비교된 행의 우로 n개 행을 더 보여 준다.
-f file	파일로부터 패턴을 입수한다.

Solaris는 등록부 /usr/xpg4/bin에 grep의 POSIX호환판본을 유지한다. 이 판번은 Linux에 의하여

리용되는 -E선택항목과 -F선택항목을 리용한다. 만일 이러한것들을 리용하려 한다면 절대경로를 리용하든지 아니면 PATH설정을 변경시켜야 한다(17.3).

패턴의 출현수(-c)

파일에 몇개의 등록부가 있겠는가? -c(count)선택항목은 패턴의 출현수를 출력한다.

```
$ grep -c 'director' emp.lst
4
```

이 지령은 emp.lst파일안에 'director'라는 패턴이 들어 있는 행의 수를 표시한다. 만일 이 지령이 여러개의 파일들을 리용하는 경우 파일이름이 행의 앞부분에 붙는다.

```
$ grep -c director emp*.lst
emp.lst:4
emp1.lst:2
emp2.lst:2
empold.lst:4
```

때때로 사용자들은 스크립트론리(script logic)에서 리용할수 있도록 모든 파일로부터 단 하나의 수값을 얻으려고 할것이다. 이와 비슷한 내용은 8.8에서 이미 론의하였으며 출력자료를 파일로서 만들기 위하여 grep지령을 리용할수 있다.

행번호의 표시(-n)

-n(number)선택항목은 패턴을 포함하는 행과 함께 행번호들을 표시하기 위하여 사용된다.

```
$ grep -n 'marketing' emp.lst
3:5678 |robert dylan    |d.g.m   |marketing |04/19/43| 85000
11:6521|derryk o'brien  |director|marketing |09/26/45|125000
14:2345|james wilcox    |g.m.    |marketing |03/12/45|110000
15:0110|julie truman    |g.m.    |marketing |12/31/40| 95000
```

행번호들은 행의 시작위치에 놓이며 두점(:)으로써 행번호를 구분한다. 여러개의 파일이름들과 함께 이 선택항목을 리용하면 2개의 마당(파일이름과 행번호)이 더 추가된다.

```
$ grep -n 'marketing' emp?.lst | head -2
emp1.lst:2:5678|robert dylan     |d.g.m    |marketing |04/19/43| 85000
emp1.lst:6:6521|derryk o'brien   |director |marketing |09/26/45|125000
```

행지우기(-v)

-v(inverse)선택항목은 패턴을 포함하는 행들을 제외한 나머지 모든 행들을 선택한다.

```
$ grep -v 'director' emp.lst > otherlst
$ wc -l otherlist
      11 otherlist                초기에 4개의 diretor가 있었다
```

이것은 행들을 지우기 위한 아주 편리한 선택항목이다. 실지로 행들은 초기파일로부터 지워 지지 않는다. 여기서는 'director'패턴을 포함한 행들을 제외한 나머지행들로 이루어 진 otherlist파일을 새롭게

만들었다.

주해

-v 선택항목은 grep의 출력으로부터 행들을 제거한다. 그러나 인수로서 지적된 파일은 변경시키지 않는다.

파일이름의 현시(-l)

-l(list)선택항목은 패턴이 발견된 파일에 대하여 그 이름만을 현시한다.

```
$ grep -l 'manager' *.lst
desig.lst
emp.lst
emp1.lst
empn.lst
```

어떤 패턴이 포함되여 있는 파일의 이름이 잘 생각나지 않는 경우 이 선택항목을 리용한다. 이것 역시 행들을 표시하지 않는다.

대소문자의 무시(i)

어떤 이름을 찾을 때 -i선택항목을 리용하면 대소문자를 구별하지 않는다.

```
$ grep -i 'WILCOX' emp.lst
2345|james wilcox    |g.m.     |marketing  |03/12/45|110000
```

우의 지령은 wilcox라는 패턴을 포함한 행들을 표시한다. 그러나 여기서는 대소문자를 구별하지 않을 뿐 패턴과 약간한 차이를 가지는 wilcocks라는 이름에 대해서는 찾지 못한다. grep는 패턴비교의 매우 세련된 기술을 제공한다.

기호 -로 시작하는 패턴(-e)

이음표 -로 시작하는 문자렬을 패턴으로 리용하면 어떻게 되겠는가? 대부분의 체계들은 아래와 같이 보여 준다.

```
$ grep "-mtime" /var/spool/cron/crontabs/*
grep: -mtime illegal option
grep[-E|-F] [-c|-l|-q] [-bhinsvx] [-e pattern_list] [-f pattern_file] [pattern_
list] [file...]
```

우에서 보여 주는바와 같이 grep는 -mtime을 자기자체의 선택항목으로서 취급하며 이러한 선택항목을 허용할수 없다는 통보를 내보낸다. 이 경우에는 -e선택항목을 리용해야 한다.

```
$ grep -e "-mtime" /var/spool/cron/crontabs/*
romeo:55 17 * * 4 find/ -name core -mtime +30 -print
```

Solaris는 등록부 /usr/xpg4/bin에 있는 POSIX호환판본에서만 이 선택항목을 제공한다. 일부 체계(특히 Linux)에서 -e선택항목은 여러개의 패턴들을 비교하도록 여러번 사용될수 있다.

Linux

기타 선택항목들

여러개의 패턴비교(-e, -f)

우에서 언급된 -e선택항목은 Linux에서 추가적인 방법으로 리용될수 있다. 이 선택항목 리용하여 단하나의 지령호출로서 여러개의 패턴을 비교할수 있다.

```
$ grep -e woodhouse -e wood -e woodcock emp.lst
2365|john woodcock   |director |personnel|05/11/47|120000
5423|barry wood      |chairman |admin    |08/30/56|160000
1265|p.j. woodhouse  |manager  |sales    |09/12/63| 90000
```

하나의 단어로 이루어 진 패턴에 대하여서는 인용부호를 붙이지 않아도 된다. 사실상 이러한 긴 지령행을 입력하는것보다 정규식을 입력하여 처리하는것이 보다 간단하다. 여기서는 이러한 표현식에 대하여 간단히 론한다. 사용자는 우와 같은 3개의 패턴을 하나의 파일에 반영하여 지령에서 이 파일을 리용하게 할수 있다. 이 경우 파일의 매개 행은 하나의 패턴에 대응된다. GNU grep는 -f선택항목으로써 파일로부터 패턴을 입수한다.

```
grep -f pattern.lst emp.lst
```

egrep와 fgrep역시 -f 선택항목을 제공하지만 서로 다른 방식으로 사용된다.

이웃행들을 인쇄하기

GNU grep는 패턴으로서 비교된 행은 물론 그 행의 우아래에 있는 행들을 보여 주기 위한 선택항목을 가지고 있다. 실례로 perl스크립트에서 사용했던 foreach명령문의 주위에 어떤 내용들이 있는가를 보기 위해서는 다음과 같은 방법을 리용한다.

```
$ grep -1 "foreach" count.pl
/print ("Region List\n ");
foreach $r_code sort (keys(%regionlist))  {
    print ("$r_code : $region{$r_code} : $regionlist{$r_code}\n);
```

이 지령은 문자렬 foreach를 탐색하여 비교된 행의 우아래로 각각 1개 행씩 표시한다. 이 선택항목은 대단히 유용하게 리용된다. 실례로 이러한 수자선택항목을 리용하여 코드토막에 포함되여 있는 유일한 문자렬로서 그 문자렬을 포함하는 코드부분을 쉽게 찾을수 있다.

사용자는 또한 -A 와 -B 선택항목을 리용하여 우아래로 각각 몇개 행씩 더 보여 주겠는가를 명확히 지정할수 있다.

```
grep -A 5 "do loop" update.sql              우로 5개 행
grep -B 3 "do loop" update.sql              아래로 3개 행
```

이 선택항목을 리용하면 비교된 행의 내용을 파악하기가 대단히 쉽다. 또한 분류된 파일들을 탐색할 때 유용하다. 이전 자료기지에서 종업원식별마당과 생년월일마당은 탐색마당으로서 자주 리용된다.

15.4 정규식(1회)

파일 emp.lst를 다시한번 보시오. 이 파일에는 trueman과 truman, wilcocks, wilcox와 같은 서로 비슷한 이름들로 되여 있는 문자렬들이 포함되여 있다. 아래의 지령은 하나의 패턴에 대하여서만 탐색한다.

```
$ grep truman emp.lst
0110|julie truman     |g.m.     |marketing |12/31/40|  95000
```

하나의 표현식으로서 서로 비슷한 파일이름들을 비교하는 쉘의 통용기호(8.2)와 같이 grep는 서로 비슷한 패턴들의 묶음을 비교하기 위하여 또 다른 부류의 표현식을 리용한다. 통용기호와는 다르게 이 표현식은 지령의 기본특징을 이루며 쉘에 대하여 하는것은 아무것도 없다. 또한 grep는 메타문자묶음(표

15-2)을 가지고 있으며 이것을 리용하면 페턴비교를 대단히 훌륭히 진행할수 있다. 어떤 표현식이 이러한 문자들을 리용하였다면 그러한 표현식을 **정규식**이라고 한다.

표 15-2. grep, sed, perl에 의하여 리용되는 정규식문자

패 턴	비 교
*	앞문자에 대하여 없거나 한번이상 출현하는것
g	문자 g에 대하여 없거나 한번이상 출현하는것
gg*	g, gg, ggg 등
.	하나의 문자를 정합한다
.*	문자에 대하여 없거나 임의의 개수의 문자
[pqr]	개별적인 문자로서 p, q, r
[abc]	a, b, c에서 어느 하나
[c1-c2]	c1과 c2에 의하여 표현되는 ASCII령역에서 임의의 문자
[1-3]	1과 3사이의 하나의 수자
[^pqr]	p, q, r문자가 아닌 하나의 문자
[^a-zA-Z]	영어자모가 아닌 문자
^pat	pat로 시작하는 행
pat$	pat로 끝나는 행
bash$	bash로 끝나는 행
^bash$	단어로서의 bash를 정합한다.
^$	아무것도 포함하지 않는 행
\{m\}	앞문자에 대하여 m번의 출현(perl에서는 기호 \을 리용하지 않는다)(15.12)
^.\{9\}nobody	행의 시작으로부터 9개 문자다음의 문자렬(nobody)을 비교(perl에서는 기호\을 리용하지 않는다)(15.12)
\{m,\}	앞문자에 대하여 최소 m번의 출현(perl에서는 기호\을 리용하지 않는다)(15.12)
\{m,n\}	앞문자에 대하여 m과 n번사이의 출현(perl에서는 기호 \을 리용하지 않는다)(15.12)
\(exp\)	\1, \2로서 참조하기 위한 표현식(perl에서는 기호 \을 리용하지 않는다)(15.12)
(BOLD\).*\1	하나의 행안에서 2개의 문자렬 BOLD(perl에서는 기호 \을 리용하지 않는다)

정규식은 일부 공통적인 질문과 치환요구를 처리한다. 즉 이러한 표현식을 리용하여 서로 비슷한 이름들에 해당한 행들을 정확히 찾을수 있다. 또한 어떤 문자렬을 지적된 문자렬로 치환할수 있으며 기호 #로 시작하는 행들을 표시할수도 있다. 그리고 지적된 렬위치에 있는 문자렬을 찾는것도 할수 있다. 이 장에서는 정규식에 대하여 3회에 걸쳐 론의한다.

정규식에서 리용되는 일부 문자들은 쉘에서도 의미를 가진다. 이러한 표현식에 대하여서는 인용부호를 붙여야 한다. 먼저 여기서는 정규식에 대하여 간단히 취급하고 앞으로 sed를 론의할 때 그 범위를 더 확장한다.

주해

정규식은 쉘에 의하여서가 아니라 지령에 의하여 해석된다. 인용부호는 쉘의 간섭이 없이 자기자체의 방식으로 메타문자들을 해석할수 있게 한다

15.4.1 문자모임

쉘의 통용기호와 같이 정규식은 꺽쇠괄호 [와]으로서 문자묶음을 둘러 싸는 문자모임을 리용한다. 패턴 비교는 묶음의 단 하나의 문자에 대하여 진행된다. 즉 표현식 [od]는 o 혹은 d의 어느 하나를 비교한다.

사용자는 또한 자모와 수자들에 대해서만 탐색하도록 그 범위를 지정할수 있다. 패턴 [a-zA-Z0-9]는 하나의 자모 혹은 수자를 정합한다. 이러한 기능은 서로 비슷한 문자렬 woodhouse와 wodehouse를 찾는데 리용할수 있다. 이 2개의 문자렬은 3번째와 4번째 문자위치에서 서로 다르다. 이와 같은 문자렬을 다 찾기 위해서는 실지로 4개의 패턴 od, oe, dd, de를 비교할수 있는 모형 [od][de]를 리용해야 한다. 우의 4개의 패턴중에서 첫번째와 네번째 패턴만이 이 문제에 해당된다. 문자모임을 리용하여 문자렬 woodhouse와 wodehouse를 찾기 위하여 요구되는 표현식은 다음과 같다.

```
wo[od][de]house
```

그러면 grep지령으로서 이러한 정규식을 리용해 보자.

```
$ grep "wo[od][de]house" emp.lst
1265|p.j. woodhouse   |manager |sales    |09/12/63| 90000
2476|jackie wodehouse |manager |sales    |05/01/59|110000
```

우의 지령은 단 하나의 패턴으로서 2개의 서로 비슷한 문자렬을 탐색하였다.

범위를 지정할 때 이음표 -의 왼쪽에 있는 문자는 오른쪽에 있는 문자보다 ASCII값으로서 보다 작아야 한다. 문자모임 [X-c]를 볼 때 X의 ASCII값은 c의 ASCII값보다 작으므로 이것은 정확하다고 말할수 있다. 그러나 표현식 [A-z]는 자모문자만을 비교하지 않는다. 즉 Z와 a사이에는 자모가 아닌 몇개의 문자들이 들어 있다.

문자모임의 반전

정규식은 문자모임을 반전시키기 위하여 기호 ^를 리용하며 쉘에서는 이러한 기능을 가진 기호로서 !가 리용된다. 문자모임이 이 기호로 시작된다면 클라스에 반영된 문자들을 제외한 나머지 모든 문자들을 비교한다. 그러므로 자모문자가 아닌 하나의 문자에 대하여 아래의 표현식을 리용할수 있다.

```
[^a-zA-Z]
```

주해

문자모임의 기능은 반전기호를 ^으로서 리용한다는점을 제외하고는 쉘에서와 비슷하다. 쉘에서는 느낌표(!)를 리용한다

15.4.2 별표(*)

별표(*)는 자기 직전의 문자를 참조한다. 여기서 말하는 별표(*)는 쉘이나 DOS에 의하여 리용되는 별표와는 본질적으로 다르다. 즉 이것은 이전 문자에 대하여 없거나 하나이상의 련속적인 문자렬을 비교한다. 다른 말로서 이전 문자는 여러번 련속적으로 나타날수 있으며 한번도 나타나지 않을수 있다. 패턴 e*은 하나의 문자 e와 이 문자로 이루어 진 문자렬을 비교한다. 또한 e문자가 전혀 나타나지 않는 경우도 고려하여 빈 문자렬도 비교한다. 비교되는 문자렬은 다음과 같다.

```
e  e  eee  eeee ....
```

e문자로 시작하는 문자렬을 비교하기 위해서는 표현식 ee*을 리용해야 한다. 통용기호에 의하여 리

용되는 별표는 이전 문자와 아무러한 관계가 없다.

주해

표현식 e*은 문자 e가 전혀 나타나지 않는 경우도 포함한다.

그러면 문자렬 trueman과 truman을 어떻게 정합하겠는가? 첫번째 패턴은 e문자를 포함하고 있지만 다른 패턴은 포함하고 있지 않다. 이것은 e문자가 나타날수 있는 경우와 나타나지 않는 경우를 념두에 둔다. 즉 이에 맞는 표현식은 다음과 같다.

```
true*man
```

우의 표현식을 리용하면 두가지 경우를 다 탐색할수 있다. 지령으로서 표현한다면 다음과 같다.

```
$ grep "ture*man" emp.lst
3564|ronie trueman    |executive |personnel |07/06/47|750000
0110|julie truman     |g.m.      |marketing |12/31/40| 95000
```

별표(*)는 문자렬만을 비교하지 않는다. 즉 표현식은 다른 패턴들을 포함할수 있는 대단히 일반적인 식이다. 우의 지령은 파일에 trueeman이라는 문자렬패턴도 있다면 이것 역시 탐색할것이다.

문자모임과 별표(*)를 리용하여 문자렬 wilcocks와 wilcox를 탐색할수 있다.

```
$ grep "wilco[cx]k*s*" emp.lst
3212|bill wilcocks     |d.g.m.    |accounts  |12/12/55| 85000
2345|james wilcox     |g.m.     |marketing |03/12/45|110000
```

표현식 k*s*은 k와 s가 나타나지 않는 경우도 포함한다. 즉 문자렬끝에 이러한 2개의 문자를 포함하지 않는 wilcox문자렬도 탐색할수 있다. 여기로부터 우리들은 정규식의 기능이 통용기호의 기능보다 우월하다는것을 알수 있다.

주해

별표(*)는 자기의 앞문자를 참조하며 그의 기능은 정규식에서만 의미를 가진다. 만일 이 문자가 정규식에서 첫 문자이라면 문자그대로서 취급된다.

15.4.3 점(.)

점(.)은 임의의 한 문자를 탐색한다. 쉘에서는 점(.)을 리용하지 않고 물음표(?)를 리용한다. 패턴 2...은 2로 시작되는 4개의 문자패턴을 탐색한다. 쉘에서는 2???로 쓴다.

정규식 .*

별표(*)와 함께 점(.)은 가장 많이 리용되는 표현식들중의 하나이다. 표현식 .*는 임의의 개수의 문자들과 빈 문자렬을 비교한다. 실례로 파일에 p.j. woodhouse라는 이름이 있는데 이에 대한 정확한 이름을 알지 못하고 그와 비슷한 이름 p.woodhouse를 알고 있다고 하자. 이 경우 탐색문자렬에 .*를 삽입하면 문제는 쉽게 해결된다.

```
$ grep "p.*woodhouse" emp.lst
1265|p.j. woodhouse    |manager  |sales     |09/12/63|  90000
```

만일 p.j. woodhouse라는 이름을 문자그대로 찾으려면 표현식은 p\.j\. woodhouse로서 되여야 할것이다. 여기서 기호 \은 점(.)을 문자그대로 취급하도록 한다.

주해

쉘에서 물음표(?)의 의미는 여기서 점(.)이 가지는 의미와 갈다.

15.4.4 패턴위치의 지정(^, $)

정규식을 리용하여 행의 시작과 끝에서 패턴을 비교할수 있다. 이러한것으로서 문자 ^와 $이 있다.

^-----행의 시작부분에서 탐색한다.

$-----행의 끝부분에서 탐색한다.

이러한 기능은 한행에 대하여 패턴이 여러번 나타날 때 자주 리용된다. 즉 해당한 위치에 지정된 패턴이 들어 있는 행들만을 탐색하려고 할 때 리용한다.

실례로서 2로 시작하는 행들만을 표시하기 위하여 간단히 2...으로서 표현식을 구성할수 있다. 그러나 이것은 수자 2와 3개의 임의의 문자로 이루어 진 문자렬만을 탐색하므로 제한이 있다. 이 경우 문자 ^을 리용하면 모든 경우를 다 표시할수 있다.

```
$ grep "^2" emp.lst
2233|charles harris    |g.m.     |sales     |12/12/52| 90000
2365|john woodcock     |director |personnel |05/11/47|120000
2476|jackie wodehouse  |manager  |sales     |05/01/59|110000
2345|james wilcox      |g.m.     |marketing |03/12/45|110000
```

같은 방법으로 기호 $을 리용하여 로임마당의 값이 70000과 89999사이의 값으로 되여 있는 행만을 표시할수 있다.

```
$ grep "[78]....$" emp.lst
5678|robert dylan     |d.g.m       |marketing |04/19/43| 85000
3212|bill wilcocks    |d.g.m.      |accounts  |12/12/55| 85000
3564|ronie trueman    |executive   |personnel |07/06/47|750000
```

그러면 탐색을 반대로 하여 2로 시작되지 않는 행들만을 선택하려면 어떻게 해야 하겠는가? 이 경우에는 표현식 ^[^2]를 리용해야 하며 지령으로서 다음과 같이 쓸수 있다.

```
grep "^[^2]" emp.lst
```

UNIX에는 등록부만을 보여 주는 지령이 없다. 그러나 관흐름을 리용하여 목록으로부터 d로서 시작되는 모든 행들을 출력한다면 문제는 쉽게 해결된다.

```
ls -l | grep "^d"                등록부만을 보여 준다
```

그러면 등록부를 보기 위해서 이와 같은 긴 부분렬을 입력해야 하겠는가? 사용자는 리용하기 편리하게 그것을 별명 혹은 쉘함수로서 변환해야 한다. 아래에는 grep가 지령 ls -l에 기능을 추가할수 있다는 것을 보여 준다. 여기서 관흐름은 묶음에서 쓰기허락을 가진 모든 파일들을 탐색한다.

```
$ ls -l | grep '^.....w'
drwxrw-r-x    3 sumit    dialout      1024 Oct 31 15:16 text
-rwxrw----    1 henry    dialout     22954 Nov  7 08:21 wall.gif
```

```
-rw-rw-r--    1 henry    dialout       717 Oct 25 09:36 wall.html
```

우의 지령렬은 ls -l지령의 출력자료로부터 6번째 렬위치의 문자 w를 탐색한다.

주해

탈자기호(^)는 정규식에서 3개의 의미를 가진다. 문자모임의 시작위치에 놓이면(실례로 [^a-z]) 클라스의 모든 문자들을 반전시킨다. 또한 문자모임의 밖에 놓이면서 표현식의 첫 위치에 놓이면(실례로 ^2....) 패턴은 행의 시작부분에서 탐색된다. 그밖의 다른 위치에서는(실례로 a^b) 문자그대로서 의미를 가진다.

15.4.5 메타문자들이 자기의 의미를 상실하는 경우

실지로 본문에는 특수문자들도 포함되여 있다. 이러한 특수문자들에 대하여 문자그대로서 비교가 진행되여야 한다면 그러한 문자들의 기능을 해제시켜야 한다. 그러나 일부 경우 즉 정규식의 규칙을 위반한 경우 자동적으로 그 기능이 해제된다. 탈자부호 ^와 같은 문자들의 의미는 표현식에서 그 문자들이 차지하는 위치에 따라 변할수 있다.

문자 -는 적당한 문자에 의하여 어느 한쪽이라도 둘러 막히지 않는 경우 문자모임에서 자기의 의미를 상실한다. 또한 묶음밖에 놓이는 경우에도 마찬가지이다. 점(.)과 별표(*)는 문자모임의 안에 놓이는 경우 자기의 의미를 상실한다. 별표(*)는 표현식의 첫 문자인 경우 문자그대로서 취급된다. 실례로 지령 grep"*"는 문자 *를 탐색한다.

그러나 패턴 g*를 탐색하려는 경우에는 별표(*)의 기능을 강제로 해제시켜야 하며 이를 위하여 문자 \을 리용한다. 실례로 문자 [을 탐색하려면 \[을 리용해야 하며 또 .*를 문자그대로서 탐색하기 위해서는 \.*를 리용해야 한다.

정규식은 UNIX체계의 어디에서나 리용된다. 즉 vi와 emacs, grep에서 리용되였으며 앞으로 egrep, sed, awk, perl, expr와 같은 강력한 UNIX지령들에도 이러한 정규식들을 리용된다. 사용자는 이러한 것들이 UNIX체계의 전문지식을 주는 기본열쇠로 되기때문에 리해해야 한다.

일부 몇개의 메타문자들에 대해서는 이 장의 뒤부분과 perl을 취급할 때에 론의한다. 그것들의 일부를 리해하기 위하여서는 먼저 egrep에 대하여 알아야 한다.

주해

정규식은 행의 시작위치로부터 가장 가까운 문자렬을 탐색한다. 또한 이 표현식은 대단히 긴 가능한 문자렬에 대하여서도 탐색이 진행된다. 그러나 표현식 03.*05는 행의 왼쪽과 오른쪽이 각각 03과 05로 둘러 막힌 행만을 표시하며 이것은 치환을 진행할 때 의의가 있다.

15.5 기타 성원들(egrep, fgrep)

egrep와 fgrep는 grep의 패턴탐색기능의 확장이다. 이 지령들은 grep의 대부분의 선택항목들을 리용하지만 자기자체의 일부 특수한 기능들을 가지고 있다. 즉 여러개의 패턴에 대해서도 능히 탐색할수 있으며 또한 파일로부터 패턴들을 입수할수도 있다.

그러면 파일로부터 패턴 woodhouse와 woodcock를 어떻게 탐색하겠는가? GNU grep지령에서는 -e선택항목을 여러번 사용하여 처리한다. 이것은 egrep지령을 리용하면 보다 더 쉬워 진다. 즉 구분문자 |을 리용하여 두 패턴사이의 경계를 준다.

```
$ egrep 'woodhouse|woodcock' emp.lst
2365|john woodcock   |director  |personnel  |05/11/47|120000
1265|p.j. woodhouse   |manager  |sales      |09/12/63| 90000
```

구분문자 |은 egrep에 의하여 리용되는 정규식문자이다. 이와 같은 문자들에 대해서는 다음절에서 론의한다. fgrep를 리용하는 경우에는 매개 패턴들을 서로 다른 행에 놓아야 한다.

```
fgrep 'woodhouse
woodcock' emp.lst
```

C쉘사용자들은 첫행의 마지막에서 기호 \을 리용하여 행바꾸기문자를 해제시켜야 한다. fgrep는 egrep에서 리용되는 구분문자 |을 비롯한 정규식문자들을 리용하지 않는다. 만일 탐색하려는 패턴이 간단한 문자렬이거나 그러한 문자렬들의 모임이라면 fgrep를 리용하는것이 더 좋다. 그것은 속도상 egrep보다 더 빠르기때문이다.

패턴을 파일에 저장하기(-f)

지금까지는 두세개의 패턴에 대하여 파일을 탐색하였다. 만일 이러한 패턴들이 많다면 어떻게 해야 하겠는가? 우의 두 지령들은 파일로부터 패턴을 입수하기 위하여 -f선택항목을 지원한다. 즉 이 경우에 모든 패턴들은 파일에 보존된다. 아래에는 egrep에서 리용되는 파일의 작성방법을 보여 준다.

```
$ cat pat.lst
admin|accounts|sales
```

fgrep에서 이 파일을 리용하기 위하여서는 다음과 같이 작성한다.

```
$ cat pat.lst
admin
accounts
sales
```

우와 같은 3개의 패턴을 탐색하기 위한 egrep와 fgrep지령은 다음과 같다.

```
egrep -f pat.lst emp.lst
fgrep -f pat.lst emp.lst
```

grep계렬의 지령들에 대한 기본결함은 마당들을 식별하기 위한 기능이 없으며 또 패턴탐색을 마당에 대해서만 진행하는것이 대단히 어렵다는것이다. 이러한 기능은 awk나 perl로서 실현한다.

egrep는 지금까지 론의된 모든 정규식문자들을 다 리용한다. 그밖에도 egrep는 일부 특수한 문자들도 리용한다. 이러한 문자들에 대해서는 정규식의 2회부분에서 론의한다.

15.6 정규식(2회)

지금까지 grep에서 리용된 정규식들은 egrep의 패턴들로서 리용될수 있다. grep와 sed지령에서 리용되는 일부 문자들은 egrep에서는 허용되지않는다. 이와 함께 egrep는 grep와 sed에서 사용되지 않는 일부 추가적인 문자들을 가지고 있다(표 15-3).

15.6.1 +와 ?

+와 ?는 egrep의 확장된 문자들이다. 이 문자들은 탐색범위를 제한하기 위하여 별표(*)가 놓이던 위치에서 자주 리용된다. +와 ?는 아래와 같은 의미를 가진다.

+------앞문자에 대하여 하나 혹은 그이상의 출현을 탐색한다.

?------앞문자에 대하여 없거나 한번의 출현을 탐색한다.

실례로 b+는 b, bb, bbb, bbbb...을 탐색한다. 즉 지정된 위치에 b가 포함되여 있지 않은 경우는 탐색하지 않는다. 표현식 b?는 한개의 b 혹은 b가 포함되여 있지 않는 경우를 탐색한다. 이러한 문자들은 별표의 기능에 비추어 볼 때 서로 비슷하다.

표 15-3. egrep와 awk에 의하여 리용되는 확장된 정규식묶음

표현식	비 교
ch+	문자 ch에 대하여 한개 혹은 그이상
g+	최소 한개의 문자g
ch?	ch문자에 대하여 없거나 한개
g?	g문자가 없거나 하나의 g문자
exp1 | exp2	표현식 exp1 혹은 exp2
GIF | JPEG	CIF 혹은 JPEG
(x1 | x2)x3	표현식 x1x3 혹은 exp2
(lock| ver)wood	lockwood 혹은 verwood

emp.lst파일에서 trueman과 truman의 두 경우만을 탐색하려면 e?형식의 표현식을 리용해야 한다.

```
$ egrep "true?man" emp.lst
3564|ronie trueman    |executive |personnel  |07/06/47|750000
0110|julie truman     |g.m.      |marketing  |12/31/40| 95000
```

문자 +는 대단히 유용한 문자이다. 사용자는 어떤 련속적인 두 단어에 대하여 탐색하려고 할 때 그 두 단어사이에 얼마만한 공백을 두고 있는지 알수 없다. 이 경우에 문자 +를 리용하면 모든 경우를 다 탐색할수 있다.

```
a□b,a□□b,a□□□b,a□□□□b, .......
```

15.6.2 다중패턴에 의한 탐색

앞절에서는 여러개의 패턴을 비교하기 위하여 구분문자 |을 리용하였다.

```
egrep 'woodhouse|woodcock' emp.lst
```

구분문자 |은 또한 egrep의 정규식묶음에서 리용되는 또 하나의 문자이다. 그외에도 괄호문자를 리용하여 패턴탐색를 더 효과적으로 할수 있다. 즉 패턴들을 하나로 묶기 위하여 괄호문자를, 패턴들을 구분하기 위하여 관(pipe)을 리용할수 있다.

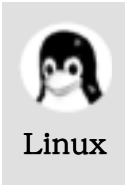
```
$ egrep 'wood(house|cock)' emp.lst
2365|john woodcock    |director |personnel |05/11/47|120000
1265|p.j. woodhouse   |manager  |sales     |09/12/63| 90000
```

또한 여기에 grep에서 사용되였던 정규식문자들을 결합시킬수 있다.

```
$ egrep 'wilco[cx]k*s*|wood(house|cock)' emp.lst
2365|john woodcock    |director  |personnel |05/11/47|120000
```

```
1265|p.j. woodhouse   |manager   |sales     |09/12/63| 90000
3212|bill wilcocks    |d.g.m.    |accounts  |12/12/55| 85000
2345|james wilcox     |g.m.      |marketing |03/12/45|110000
```

egrep는 grep에서 리용되는 정규식들을 다 지원하는것은 아니다. 표 15-2에는 egrep에 의하여 지원되지 않는 일부 문자들과 실례들이 있다. 이러한것들에 대해서는 sed를 취급할 때 론의한다.

Linux

사용자는 egrep의 확장된 정규식들을 리용하기 위하여 egrep -E를 리용할수 있다. -F선택항목은 grep가 fgrep와 같이 동작하도록 한다.

15.7 흐름편집기(sed)

sed는 려파기능들을 결합한 여러가지 목적에 리용되는 하나의 도구이다. 이 도구는 리 맥몬(Lee McMahon)에 의하여 설계되였으며 초기의 UNIX행편집기 ed에 시초를 두고 있다. sed는 호상작용이 없는 조작들을 수행하기 위하여 사용되며 자료흐름에서 동작한다.

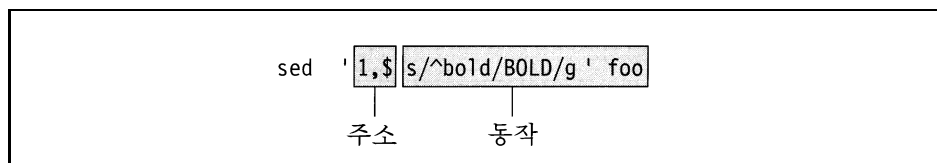
sed는 몇개의 선택항목들을 가지고 있다. 이러한 선택항목들의 기능은 행들을 선택하고 선택된 행들에 대하여 적용할 명령들을 작성할수 있는 편리로부터 나왔다. 그것은 또한 프로그람작성언어와 구별되는 여러가지 특징을 가지고 있다. 이러한 기능들은 perl을 론할 때 취급되였기때문에 여기서는 그와의 경계를 명백히 하기 위하여 간단히 서술한다. 사실상 perl은 우와 같은 기능들을 대단히 능숙히 처리한다.

sed에서 리용되는 모든것은 어떤 **명령**(instruction)이다. 명령은 행들을 선택하기 위한 **주소**(address)와 그에 대한 **동작**(action)으로 이루어 져 있다.

```
sed 선택항목들 '주소 동작' 파일1 파일2 파일3.....
```

주소와 동작은 외인용부호안에 놓인다. 동작요소는 sed계렬의 내부지령으로서 표현된다(표 15-4). 내부지령으로서 본문의 삽입, 삭제, 수정과 같은 처리를 간단히 진행할수 있다. 그림 15-1은 sed명령에 대한 구성요소들을 보여 준다. 사용자는 하나의 sed지령에 여러개의 명령들을 놓을수 있으며 이것은 보다 더 성능이 강한 지령으로 되게 한다.

그림 15-1. sed명령의 구성요소

주해

C쉘사용자들이 sed지령을 다음행에 련속적으로 쓰기 위하여 [Enter]건을 누르면 쉘은 오유통보문을 내보낸다. 이 경우에는 지령의 마지막행을 제외한 매행들에 대하여 그끝에 사선기호 \를 리용하여 지령의 실행을 금지시켜야 한다. 이 상태에서 [Enter]건을 누르면 ?프롬프트가 발생된다. Solaris와 같은 일부체계들은 이러한 프롬프트를 표시하지 않는다. 이러한 해제기능은 항시적으로 적용되는것이 아니라 일부 경우에만 적용된다. 즉 이 기능이 어떤 경우에는 동작하지 않는다. 이것은 현재의 쉘이 그러한 형식의 지령을 해석하지 못한다는것을 의미한다.

제17장에서는 Bourne쉘과 C쉘에 비한 Korn쉘과 bash쉘의 우월성에 대하여 보여 주고 있다. 즉 가입쉘로서 Korn 혹은 bash를 리용하면 sed지령을 리용하기가 더 편리하다는것을 알수 있

다. 사용자는 자기의 가입쉘을 변경시키지 않고도 sed지령과 다음장에서 론의되는 awk프로그람들을 실행시키기 위한 다른 쉘을 리용할수 있다. 이를 위해서는 간단히 sh, ksh 혹은 bash지령을 실행시키시오(이 세 지령은 체계상에서 어느때나 유효하다). 이 상태에서 작업은 계속된다. 마지막에 자기의 가입쉘로 귀환하기 위해서는 exit지령을 리용해야 한다.

표 15-4. sed에 의하여 리용되는 내부지령

지 령	의 미
i,a,c	본문의 삽입, 추가, 변경
d	행지우기
1,4d	첫번째 행부터 4번째 행까지의 지우기
r for	행다음에 파일 foo의 내용을 놓기
w bar	파일 bar에 지적된 행의 쓰기
p	표준출력장치에 행들을 출력하기
3,$p	3행부터 나머지 모든 행들을 출력한다(-n선택항목이 요구됨)
$!p	마지막행을 제외한 나머지 모든 행들을 출력한다(-n선택항목이 요구됨)
/begin/,/end/p	begin과 end로 둘러 막힌 행들을 출력한다(-n선택항목이 요구됨)
q	지적된 행을 읽고 중지한다
10q	처음 10개 행을 읽고 중지한다
=	주소화된 행수를 출력한다
s/s1/s2	모든 행에 대하여 처음으로 나타나는 문자렬 s1 혹은 표현식 s2에 맞는 문자렬을 s2이라는 문자렬로 치환한다
10,20s/-/:/	10행부터 20행사이에 첫번째로 나타나는 문자 -를 ':'으로 교체한다
s/s1/s2/g	모든 행에 대하여 문자렬 s1 혹은 표현식 s1에 맞는 문자렬들을 s2이라는 문자렬로 치환한다
s/-/:/g	모든 행에 대하여 문자 -를 :로 치환한다

15.8 행주소화

sed에서 행주소화는 두가지 방법으로 진행된다.

- 행번호에 의한 주소화(실례 3,7p)
- 패턴에 의한 주소화(실례 /From:/p)

첫번째 형식에서 주소는 한개의 행을 지적하기 위하여 하나의 행번호를 지적할수 있으며 서로 린접한 행들의 묶음을 선택하기 위하여 행번호로서 범위를 지적할수 있다. 어느 경우에나 동작(출력지령 p)은 이러한 주소다음에 놓인다.

그러면 실례로서 먼저 행번호에 의한 주소화방법을 리용한 명령 3q에 대하여 보자. 이 명령은 주소 3과 동작 q로써 갈라 볼수 있다. 아래의 지령은 head -3지령과 같은 결과를 가져 온다.

```
$ sed '3q' emp.lst
2233|charles harris   |g.m.     |sales     |12/12/52| 90000
9876|bill johnson     |director |production|03/12/50|130000
5678|robert dylan     |d.g.m    |marketing |04/19/43| 85000
```

sed는 또한 출력자료를 화면에 현시하기 위하여 p지령을 리용한다. 그러면 p지령으로서 두개의 행주소를 리용하면 어떻게 되는가를 보시오.

```
$ sed '1,2p' emp.lst
2233|charles harris    |g.m.      |sales      |12/12/52| 90000
```

```
2233|charles harris    |g.m.      |sales      |12/12/52| 90000
9876|bill johnson      |director  |production |03/12/50|130000
9876|bill johnson      |director  |production |03/12/50|130000
5678|robert dylan      |d.g.m     |marketing  |04/19/43| 85000
2365|john woodcock     |director  |personnel  |05/11/47|120000
...more lines with each line displayed only once...
```

기정적으로 sed는 동작(action)에 의하여 영향을 받는 행외에도 표준출력장치에 모든 행들을 표시한다. 또한 주소화된 행들을 화면에 두번 반복하여 표시한다.

우와 같은 출력을 방지하기 위하여서는 sed와 함께 선택항목을 리용해야 한다. 이러한 문제에 대해서는 다음에 론한다.

중복행출력의 금지(-n)

행의 반복출력을 없애기 위해서는 p지령을 리용할 때마다 -n선택항목을 리용해야 한다. 우의 지령을 다시 쓰면 다음과 같다.

```
$ sed -n '1,2p' emp.lst
2233|charles harris   |g.m.      |sales      |12/12/52| 90000
9876|bill johnson     |director  |production |03/12/50|130000
```

파일에서 마지막행을 선택하기 위해서는 기호 $을 리용해야 한다.

```
$ sed -n '$p' emp.lst
0110|julie truman     |g.m.     |marketing  |12/31/40| 95000
```

주소와 동작은 보통 외인용부호안에 놓인다. 그러나 sed명령에 파라메터값이나 지령치환이 포함되여 있는 경우에는 겹인용부호를 리용해야 한다.

행선택기준의 반전(!)

사용자는 동작(action)과 함께 sed의 반전연산자 !를 리용할수 있다. 첫 두행을 선택하는것은 그 두 행을 제외한 나머지 모든 행들을 선택하지 않는것과 같다. 이에 대한 지령을 다음과 같이 쓸수 있다.

```
sed -n '3,$!p' emp.lst              3번째 행부터 마지막행까지 표시하지 않는다
```

임의의 위치에서의 행들의 선택

sed는 파일의 임의의 위치로부터 행들을 선택할수 있다. 이것은 head나 tail지령으로써는 불가능하다.

```
sed -n '9,11p' emp.lst              9행부터 11행까지
```

여러개의 단락들의 선택

sed지령으로써 행들의 련속적인 묶음들을 선택할수 있다. 즉 서로 다른 행에 개개의 명령을 놓는것으로써 많은 단락들을 선택할수 있다.

```
sed -n '1,2,p                하나의 행에서 3개의 주소
7,9p
$p' emp.lst                 마지막행을 선택한다
```

또한 한행에 모든 명령들을 다 놓을수 있다. 이 경우에는 -e선택항목을 리용해야 한다.

```
sed -n -e '1,2p' -e '7,9p' -e '$p' emp.lst              우의 지령과 같다
```

참고

한행에 대하여 두번 반복되여 표시되는것을 피하기 위하여서는 p지령을 리용할 때마다 -n선택항목을 함께 리용해야 한다.

15.9 문맥주소화

주소화의 두번째 형식은 행번호가 아니라 하나 혹은 두개의 패턴을 지적하는것이며 이것을 **문맥주소화**(context addressing)라고 한다. 패턴에 대해서는 사선기호 /로 둘러 막아야 한다. 이 형식을 리용한다면 우편함($HOME/mbox)에서 발송자들을 쉽게 탐색할수 있다.

```
$ sed -n '/From: /p' $HOME/mbox
Form: janis joplin <joplinj@altavista.net>
From: charles king <charlesk@rocketmail.com>
From: Monica Johnson <Monicaj@Web6000.com>
From: The Economist <business@lists.economist.com>
```

awk나 perl 역시 이러한 형식을 지원한다. 우의 실례에서는 From:으로 시작되는 모든 행들을 표시하였다. sed는 또한 grep에서 리용한 정규식들을 허용한다. 아래의 지령들은 사용자들의 기억을 되살린다.

```
sed -n '/^From: /p' $HOME/mbox              ^은 행의 시작위치에서 비교한다
sed -n '/wilco[cx]k*s*/p' emp.lst              wilcox와 wilcocks
sed -n "/o'br[iy][ae]n/p              o'briens 혹은 lennon
/lennon/p" emp.lst
```

3번째 지령에서는 패턴 그자체가 외인용부호를 포함하고 있기때문에 겹인용부호를 리용하였다. 겹인용부호가 외인용부호를 보호하듯이 외인용부호도 겹인용부호를 보호한다.

C쉘

C쉘사용자들은 우의 3번째 실례를 리용하는 경우 첫행의 마지막에 사선기호 \ 를 추가해야 한다. 그렇지 않으면 겹 혹은 외인용부호가 빠졌다는 오유통보문 Unmatched ".을 내보낸다.

또한 사용자는 련속적인 행들의 묶음을 선택하기 위해서 반점으로서 구별되는 여러개의 문맥주소들을 지적할수 있으며 행주소와 문맥주소의 혼합으로서도 지적할수도 있다.

```
sed -n '/johnson/,/lightfoot/p' emp.lst
sed -n '1, /woodcock/p' emp.lst              반점다음에 공백이 있다
```

앞의 실례(15.4)에서는 쓰기허락을 가진 파일들을 보기 위하여 관흐름으로써 ls지령과 grep지령을 결합하여 리용하였다. 이것은 grep대신 sed지령과 결합하여 쓸수도 있다.

```
ls -l | sed -n '/^.....w/p'
```

grep와 sed에서 리용한 표현식들은 우리가 지금까지 본 표현식들보다 기능이 더 강하다. 앞으로 이러한 표현식들은 몇개의 특수문자들을 더 리용하며 이 장에서 취급되는 정규식의 3회부분에서 더 론한다.

주해

모든 문맥주소들은 사선기호 /로 둘러 막아야 한다. 또한 grep에서 리용한 모든 표현식들을 문맥에 포함시킬수 있다. 그러나 egrep형의 표현식들은 포함시킬수 없다.

15.10 본문편집

행들을 선택하는것외에도 sed는 본문편집기능도 가지고 있다. vi와 같은 방식으로서 sed는 i(삽입), d(추가), c(변경), r(읽기)지령들을 리용한다. 그러면 이러한 지령들에 대하여 하나하나 보기로 하자.

15.10.1. 본문의 삽입과 변경(i, a, c)

본문을 추가하기 위해서는 a지령을 리용해야 하며 자기가 요구하는 행들을 입력해야 한다. 이 경우 지령의 마지막행을 제외한 나머지행들의 끝에 기호 \을 추가해야 한다. 이 지령으로서 perl서고의 마지막부분에 아래와 같은 두행을 추가할수 있다.

```
$ sed '$s\                    파일의 끝에 추가
> # You must place the following line at the end\
> 1;
> ' www_lib.pl > $$
```

우에서 보는바와 같이 본문을 입력한 다음 [Enter]건을 누르기에 앞서 행끝에 기호 \을 추가해야 한다. 이러한 기술은 i와 c지령을 리용할 때도 마찬가지이다. 쉘의 PID를 나타내는 기호 $$은 수자를 리용한 파일이름을 달기 위하여 사용된다. 사용자는 여기서 자기의 요구에 맞는 임의의 파일이름을 리용할 수 있다. 즉 기호 $$이 리용되면 이미 존재하는 파일에 대하여 덧쓰기하지 않는다.

두줄공간형식의 본문

아래의 지령과 같이 주소를 리용하지 않으면 어떤 결과를 초래하겠는가? 삽입 혹은 수정될 본문은 파일의 매행에 대하여 앞뒤에 놓인다. 지령

```
sed 'i\                    매행앞에 삽입
                           빈 행
' foo
```

은 파일의 매행이 출력되기전에 빈 행을 삽입한다. 이것은 본문을 두줄공간화하기 위한 또 하나의 방법이다(9.4). i와 a지령의 차이점은 하나는 본문을 지적된 행의 앞에, 다른 하나는 뒤에 삽입한다는것이다.

C쉘

우의 지령은 C쉘에서는 동작하지 않는다. C쉘에서 동작시키기는 경우 하나의 사선기호에 대해서는 2개의 사선기호, 없는 경우에는 하나의 사선기호를 더 써주어야 한다. 우의 지령은 C쉘에서 다음과 같이 쓸수 있다.

```
sed 'i\\                   두개의 사선기호
\                 하나의 사선기호
' foo
```

이것은 사용하기가 불편한 형식이며 그리 직관적이지 못하다. sed, awk, perl지령들은 다른 쉘에서 실행될수 있다.

파일에서 읽기(r)

r지령은 파일의 지정된 위치에서 어떤 파일의 내용을 읽는다. 아래의 실례는 파일 form_entry.html의 <FORM>끄리표다음에 외부파일 template.html의 내용을 삽입하는 방법을 보여 준다.

```
sed '/<FORM>/r template.html' form_entry.html
```

15.10.2. 행의 삭제

d지령을 리용하면 sed는 패턴이 포함되여 있지 않는 행들을 선택하기 위한 grep의 -v선택항목과 비슷한 동작을 수행한다. 이 지령을 리용하면 쉘 혹은 perl스크립트의 설명문들을 제거할수 있다.

```
sed '/^#/d' foo > bar
sed -n '/^#/!p' foo > bar
```

빈 행의 제거

빈 행은 임의의 개수의 공간들과 타브들로 이루어 져 있다. 그러면 파일로부터 이러한 공백행들을 어떻게 지우겠는가? 이를 위해서는 공간문자와 타브문자를 검출하기 위한 패턴을 작성해야 한다.

```
sed '/^[□        ]*$/d' foo                하나의 공간과 타브
```

이 실례에서 보는바와 같이 공간다음에 [Tab]건 혹은 [Ctrl-i]건을 눌러야 한다. 시작과 끝부분에 문자 ^와 $을 리용하는것은 그 어떤 내용도 포함하지 않는 행들만을 탐색하기 위해서이다.

15.10.3. 파일에 쓰기(w)

w(write)지령은 파일에 선택된 행들을 쓴다. 아래의 실례는 끄리표 <FORM>과 </FORM>사이의 행들을 어떤 개별적인 파일로 보관한다.

```
sed '/<FORM>/,/<\/FORM>/w form.html' pricelist.html
```

HTML파일에서 끄리표 <FORM>은 자기와 대응되는 </FORM>끄리표를 가지고 있다. 문자 /은 sed의 패턴구분문자이기때문에 이 문자의 기능을 해제시켜야 한다. 여기서는 form내용들이 파일 forms.html로 보존된다. 아래의 지령과 같이 사용자는 여러개의 HTML파일들의 form토막들을 하나의 파일로써 보존할수 있다.

```
sed '/<FORM>/,/<\/FORM>/w forms.html' *.html
```

sed에 대해서는 앞으로 가면서 더 론의한다. 아래의 실례는 3개의 패턴을 탐색하며 매개의 패턴에 의하여 선택된 행들을 서로 다른 파일에 보관한다.

```
sed '/<FORM>/,/<\/FORM>/w forms.html
    /<FRAME>/,/<\/FRAME>/w frames.html
    /<TABLE>/,/<\/TABLE>/w tables.html' pricelist.html
```

주해 w지령은 개별적인 파일들에 씌여 질 행들에 관계없이 마지막에 모든 행들을 출력한다. 이것을 피하기 위해서는 -n선택항목을 리용하시오.

참고

파일로부터 명령들을 받아 들이자면 -f선택항목을 리용하시오. 우의 지령을 고쳐서 쓰면 다음과 같다.

sed -f instr.fil emp.lst

여기서 instr.fil파일은 아래와 같은 형식의 명령들을 포함하고 있다.

```
/<FORM>/,/<\/FORM>/w forms.html
/<FRAME>/,/<\/FRAME>/w frames.html
/<TABLE>/,/<\/TABLE>/w tables.html
```

이 지령은 또한 -e선택항목을 리용하여 명령들을 한 행에 쓸수도 있다.

15.11 치환

sed의 강력한 특징의 하나는 치환기능이며 이것은 s(substitute)지령으로 실현한다. 이 지령은 입력자료에서 어떤 패턴을 그밖의 다른것으로써 치환한다. 이전에(4.16) vi에서 아래와 같은 문법을 취급하였다.

```
[주소]s/표현식1/문자렬2/기발
```

여기서 표현식1(정규식일수도 있다.)은 [주소]에 의하여 선택된 모든 행에서 문자렬 2로 치환된다. vi에서와는 다르게 주소가 지정되지 않으면 치환은 모든 행에 대하여 진행된다. 아래의 실례는 문자 |을 두점 :으로 치환한다.

```
$ sed 's/|/:/' emp.lst | head -2
2233:charles harris    |g.m.     |sales      |12/12/52| 90000
9876:bill johnson      |director |production |03/12/50|130000
```

우에서 보여 주는바와 같이 지령은 치환을 행에서 문자 |의 첫 실체에 대하여서만 진행하였다. 모든 관흐름들을 치환하기 위해서는 g(global)기발을 리용하여야 한다.

```
$ sed 's/|/:/g' emp.lst | head -2
2233:charles harris    :g.m.      :sales      :12/12/52: 90000
9876:bill johnson      :director  :production :03/12/50:130000
```

실례에서는 두점 :으로서 모든 관흐름을 치환하기 위하여 대역치환을 진행하였다. 여기서는 두개 행만을 보여 주었지만 치환은 파일전체에 대하여 진행되였다. 사용자는 주소를 지정하는것으로써 치환범위를 제한할수 있다.

```
sed '1,3s/|/:/g' emp.lst                처음 3개의 행에 대해서만 치환이 진행된다
```

치환은 개별적인 문자에 대해서는 적용할수 없으며 임의의 문자렬이 될수 있다. 치환될 문자렬은 정규식으로도 표현할수 있다.

```
sed 's/<I>/<EM>/g' foo.html
sed -n 's/gilmo[ur][re]/gilmour/p' emp.lst
```

우의 2번째 실례에서는 -n선택항목을 리용하여 문자렬 gilmour으로서 gilmour와 gilmore를 치환할뿐만아니라 그러한 행들을 선택한다.

치환이 정확히 진행되였는가를 검사하기

sed는 화면에 파일전체의 내용을 표시하며 이때 사용자는 치환이 정확히 진행되였는가 하는것을 알수 없다. grep와는 다르게 패턴을 찾을수 없는 경우 sed는 실패로 인정하지 않는다.

그러면 치환이 정확히 진행되였는가를 어떻게 알수 있겠는가? UNIX의 다른 려파기능을 리용하면 sed지령에 의하여 치환된 관(pipe)들의 수를 표시할수 있다.

```
$ sed 's/|/:/g' emp.lst | cmp -l - emp.lst | wc -l
        75
```

실례에서 sed의 출력자료는 초기의 파일과 비교된다(cmp의 -l선택항목은 비교되지 않는 문자에 대한 행들을 따로 묶는다). wc는 이러한 행들을 계수하며 75개 관이 치환되였다는것을 통보한다. 수값 0은 치환할 패턴이 없다는것을 의미한다.

여러개의 치환을 동시에 진행하기

sed에 대한 한번의 호출로써 여러개의 치환을 동시에 진행할수 있다. 즉 매 명령의 끝에서 [Enter]건을 누르시오. 그다음 마지막에 인용부호를 닫으시오.

```
$ sed 's/<I>/<EM>/g              csh에 대해서는 마지막행을
> s/<B>/<STRONG>/g               제외한 모든 행의
> s/<U>/<EM>/g' form.html        끝에 \기호를 추가하시오
```

sed는 하나의 흐름편집기이다. 즉 자료흐름에서 동작한다. 이것은 어떤 명령이 자기의 이전 명령의 출력자료를 처리한다는것을 의미한다. 사람들은 흔히 이러한 내용을 잊어 버린다. 아래의 지령렬에서는 결과적으로 <I>꼬리표들이 <STRONG>으로 바뀐다.

```
$ sed 's/<I>/<EM>/g
> s/<EM>/<STRONG>/g' form.html
```

어떤 명령묶음을 실행하는 경우 파일에 이러한 s명령들을 보관할수 있으며 이때 sed의 -f선택항목을 리용해야 한다.

주해 명령의 끝에 g가 있는 경우 치환은 행에 대하여 전체적으로 진행된다. 만일 g가 없으면 행의 가장 왼쪽에 있는것만 치환된다.

여러개의 공간을 압축하기

종업원자료기지의 두번째, 세번째, 네번째 마당으로부터 공간을 어떻게 제거하겠는가? 이를 위해서는 원천문자렬로서 요구되는 표현식이 공백문자들을 나타내야 한다. 아래의 실례는 마당의 뒤부분에 있는 공간만을 제거한다.

```
$ sed 's^ *|^|^g' emp.lst | head -2           별표전에 공백이 있다
2233|charles harris|g.m.    |sales     |12/12/52| 90000
9876|bill johnson  |director|production|03/12/50|130000
```

여기서는 문자 /대신에 ^을 리용하였다. sed(와 vi)는 문자 ^가 문자렬에서 나타나지 않는한 패턴구분문자로서 리용한다. 대부분의 UNIX체계파일들은(/etc/pas swd와 같이) UNIX도구들이 구분문자로 마

당들을 식별하기때문에 가변길이형식을 가진다. 이것은 후에 awk지령에서 리용되는 파일형식이다.

주해

-n선택항목과 출력(p)지령은 일반적으로 치환을 진행하기 위하여 사용되지 않는다. 이러한것들은 치환이 진행되였든 안되였든 상관없이 모든 행들을 출력한다.

15.11.1 기억패턴

지금까지는 패턴을 탐색하여 그것을 다른것으로써 치환하는 방법에 대하여 보았다. 아래의 지령들은 정확히 말하면 다 같은 처리를 진행한다.

```
sed 's/director/member/' emp.lst
sed '/director/s//member/' emp.lst
sed '/director/s/director/member/' emp.lst
```

우의 실례에서 두번째 형식은 sed지령이 탐색된 패턴을 기억하고 있으며 두개의 사선기호안에 그것을 보존하고 있다고 말할수 있다. 여기서 빈 표현식을 나타내는 2개의 사선기호는 탐색된 패턴과 치환될 패턴이 같다는것을 의미한다. 여기서는 이러한것을 **기억패턴**(remembered pathern)이라고 한다.

목적문자렬로서 //을 리용하면 전체적으로 원천패턴을 제거한다는것을 의미한다.

```
sed 's/|//g' emp.lst                파일로부터 모든 관흐름을 제거한다
```

세번째 형식에서 주소 /director/는 중복되여 나타난다. 그러나 치환령역을 넓히기 위해서는 이러한 형식을 리해해야 한다. 이 방법을 리용하면 어떤 문자렬이 들어 있는 모든 행들에서 필요한 문자렬을 다른것으로 치환할수 있다.

```
$ sed -n '/marketing/s/director/member/p' emp.lst
6521|derryk o'brien    |director   |marketing  |09/26/45|125000
```

주해

두 사선기호의 의미는 명령에서 그 위치에 관계된다. 그것이 원천문자렬에 있으면 탐색된 패턴이 그 위치에 보관된다는것을 의미한다. 또한 목적문자렬에 //이 놓이면 원천패턴을 제거한다는 것을 의미한다.

15.11.2 반복패턴

원천문자렬에서 패턴이 치환될 문자렬에서 반복되여 나타나는 경우 그것이 나타나는 위치에 특별히 문자 &를 리용할수 있다. 아래의 모든 지령들은 다 같은 처리를 진행한다.

```
sed 's/director/executive direcotr/' emp.lst
sed 's/director/executive &/' emp.lst
sed '/director/s//executive &/' emp.lst
```

문자 &는 **반복패턴**(repeated pathern)이라고 말하며 이것은 원천문자렬전체를 대신한다. 수자꼬리표에 대해서는 후에 론하기로 하고 여기서 문자 &는 치환문자렬에서 리용할수 있는 특수한 문자이다. 다른 모든 문자들은 문자그대로 취급된다.

15.12 정규식(3회)

아직까지는 정규식묶음에 대하여 다 취급하지 않았다. 즉 grep와 sed에서 리용되는 문자들이 더 있다. 여기서는 매개 메타문자들앞에 리용되는 문자 \에 대하여 본다. 앞으로 여기서는 표현식의 두가지 형태에 대하여 배운다.

- 구간정규식 - 이 표현식은 괄호 { }를 리용하며 그사이에 하나 혹은 한조의 수자들이 놓인다.
- 꼬리표 붙은 정규식- 이 표현식은 괄호 ( )로서 패턴들을 묶는다.

두 표현식에서는 지령이 메타문자들을 특별한 의미로 리해하도록 문자 \을 리용한다. 쉘은 표현식 그 자체가 인용괄호안에 포함되여 있기때문에 그에 대하여 아무러한 조작도 하지 않는다.

15.12.1 구간정규식(IRE)

지금까지는 행의 시작과 끝위치에서 패턴을 정합하였다. 그러면 임의의 지정된 위치 혹은 어떤 구역안에서 패턴을 정합할수 없겠는가? sed와 grep는 특수한 형식의 정규식을 받아 들인다. 즉 이 정규식은 패턴앞에 문자들의 수를 지적하는 옹근수값을 리용한다.

실례로 어떤 목록에 대하여 생각해 보자. 이 목록에서 쓰기허락비트들은 3, 4, 9번째 문자위치를 차지한다. grep로 쓰기허락설정을 가진 파일들을 표시하기 위해서는 다음과 같이 할수 있다.

```
$ ls -l | grep "^.\{5\}w"
-r-xrw-r-x    1 sumit     dialout      527 Apr 23 07:42 valcode.sh
-r-xrw-r-x    2 sumit     dialout      289 Apr 23 07:42 vvi.sh
```

그러면 먼저 표현식 ^.\{5\}w에 대하여 분석해 보자. 이 표현식은 행의 앞으로부터 임의의 5개 문자와 w문자를 결합한 문자렬에 대하여 탐색한다. 표현식 \{5\}은 앞문자가 5번 나타나야 한다는것을 의미한다. \에 의해 의미해제된 괄호를 리용한 이 표현식을 여기서는 **구간정규식**(IRC:interal regular expresion)이라고 부른다. 이것은 3가지 형식을 가지고 있다.

- ch\{m\} - 여기서는 메타문자 ch가 m번 나타난다.
- ch\{m,n\} - 문자 ch는 m과 n사이의 수값만큼 나타난다.
- ch\{m,\}- 문자 ch는 최소 m번 나타날수 있다.

이러한 모든 형식들은 첫 요소로서 단일문자표현식 ch를 가지고 있다. 이것은 임의의 문자, 점(.), 문자모임의 어느하나일수 있다. 그뒤로는 괄호 { }로 둘러 막힌 하나의 수값 m 혹은 ch문자가 나타나는 개수를 결정하기 위한 m부터 n까지의 범위가 놓인다. m과 n은 255이하의 수값이여야 한다.

그러면 IRE의 두번째 형식에 대하여 보자. 이것은 어떤 범위안에서 패턴을 탐색할수 있기때문에 쓰기비트가 설정된 파일에 대하여 화면에 표시할수 있다.

```
$ ls -l | grep "^.\{5,8\}w"
-r-xr-xrwx    3 sumit     dialout      426 Feb 26 19:58 comj
-r-xr-xrwx    3 sumit     dialout      426 Feb 26 19.58 runj
-r-xrw-r-x    1 sumit     dialout      527 Apr 23 07:42 valcode.sh
-r-xrw-r-x    2 sumit     dialout      289 Apr 23 07:42 vvi.sh
```

이전 실례(15.4)에서는 행에서 어떤 위치에 있는 패턴을 찾기 위해서 다섯개의 점을 리용하였다. 그

러면 이러한 문자들이 100개 있는 경우에는 어떻게 해야 하겠는가? 실례자료기지에서 년도는 50렬부터 시작된다. 아래의 지령을 리용하면 이 자료기지에서 1945년도에 태여난 사람들을 쉽게 찾을수 있다.

```
$ sed -n '/^.\{49\}45/p' emp.lst
6521|derryk o'brien    |director   |marketing  |09/26/45|125000
2345|james wilcox      |g.m.       |marketing  |03/12/45|110000
```

길이에 의한 행선택

IRE를 리용하여 길이가 100이상인 행들을 선택할수 있다. 아래의 두번째 지령은 최대길이를 150으로 제한하였다.

```
sed -n '/.\{101,\}/p' foo               길이가 최소 101인 행
grep '^.\{101,150\}$' foo               길이가 101부터 105사이에 놓이는 행
```

두번째 실례에서 메타문자 ^와 $은 행의 길이가 150이상인것들을 선택하지 않기 위하여 리용한다. 정규식은 가장 긴 패턴을 가능한껏 탐색해 본다.

15.12.2 꼬리표 붙은 정규식(TRE)

먼저 sed에 대하여 이 특징을 론하자. 실생활에서는 grep보다도 sed지령에 이 표현식을 많이 리용한다.

sed는 원천문자렬전체를 복사하기 위하여 &를 리용한다. 그러나 문자렬의 부분도 복사할수 있다. 만일 괄호로써 원천문자렬안의 패턴들을 개별적으로 묶는다면 sed는 거기에 수자꼬리표를 붙인다. 이 꼬리표는 반복되는 패턴에 대하여 임의의 위치에서 사용할수 있다.

실례로 행의 첫 묶음으로서 패턴 \ (higgins\ )은 정확히 말하면 문자렬 higgins로서 리해한다. sed는 그것을 \ 1로 기억하고 생략해 버린다. 이것은 원천문자렬과 목적문자렬에서 higgins는 \ 1로 표현될수 있다는것을 의미한다. 이러한 꼬리표들은 \ 2, \ 3 등으로 표현된다. 그러면 간단한 실례로서 이것을 리해해 보자. 즉 본문에서 문자렬 henry higgins를 higgins, henry로 치환하자. sed치환명령은 다음과 같다.

```
$ echo "henry higgins" | sed 's/\(henry\) \(higgins\)/\2,\1/'
higgins, henry
```

원천문자렬에는 꼬리표가 리용될 패턴 \(henry\)와 \(higgins\)이 놓여 있다. 그것들은 수자꼬리표 \1과 \2로서 원천문자렬에 복사된다. 이러한 매개 패턴들을 **꼬리표 붙은 정규식**(TRE:tagged regular expression)이라고 한다. (,), 1, 2는 sed가 특별히 취급하도록 문자 \으로써 해제시켜야 한다. 반점은 문자그대로서 취급된다. 그러므로 두 패턴사이에 반점이 놓인다.

반복되는 단어의 탐색

문서작성자들에게 도움이 되는 또 하나의 실례를 생각해 보자. TRE는 the the와 같은 반복되는 단어들을 탐색하는 기능을 가지고 있다. TRE는 묶음화된 패턴을 기억하고 있기때문에 아래와 같은 반복되는 단어들을 쉽게 찾을수 있다.

```
$ grep "\([a-z][a-z][a-z]*\)□□*\1" note          별표(*)앞에 두개의 공백이 있다
You search search for a pattern with grep.
sed    sed can perform substitution too.
```

```
But the grand-daddy of them all is perl perl.
```

모든 행들에는 반복되는 패턴이 들어 있다. 그러면 우의 지령은 무엇을 비교하는가? 지령은 소문자로서 최소 두개의 문자로 이루어 진 단어들을 탐색한다. 이 묶음뒤에는 하나 혹은 그이상의 공백들과(여기서는 두개의 공백) 반복되는 패턴이 놓인다.

우편주소의 생성

emp.lst(15.1)에서 모든 사람들이 마지막이름뒤에 령역 planets.com이 붙은 사용자등록자리를 가지고 있다고 생각하자. 그러면 이러한 등록자리들이 쉘스크립트에서 사용될수 있도록 전자우편주소목록을 생성할수 있겠는가? TRE를 리용하면 간단히 처리할수 있다. 즉 cut지령으로서 먼저 두번째 마당을 선택하고 매개 묶음이 하나의 단어를 나타내도록 두개의 묶음을 형성한다. 이 단어는 공백이 아닌 문자들로서 이루어 졌다고 보아야 한다.

```
$ cut -d\| -f2 emp.lst | sed 's/\([^ ]*\) \([^ ]*\)/\2@planets.com/'
harris@planets.com
johnson@planets.com
dylan@planets.com
woodcock@planets.com
  ……
```

첫번째 이름과 마지막이름은 TRE \([^]*\)으로써 충분히 표현된다. 목적문자렬은 마지막이름(\2)과 문자렬 @planets으로 이루어 진다.

URL에서 경로이름의 바꾸기

마지막으로 TRE의 중요한 응용에 대하여 보자. HTML문서에서 그와 련결된 문서가 현재의 등록부에 있다 해도 화면에 표시되지 않는 일련의 문제가 제기된다. 이것은 문서의 위치를 지정하는 경로들이 련결꼬리표들에서 수정되여 있지 않기때문이다(14.12). 실례로서 아래와 같은 목록을 생각해 보자.

```
$ cat httplinks.html
<LI><A HREF="smail.html">Sendmail</A> The Universal Mail Transport Agent
<LI><A HREF="http://www.sonu.com/docs/ftpdoc.html"File Transfer Protocol</A>
<LI><A HREF="../../public_html/news.html">Newsgroups</A> Usenet News
<LI><A HREF="../irc.html">Internet Relay Chat</A> On-Line text conversation
```

마지막으로부터 3개의 항목들은 등록부를 리용한 경로를 가지고 있으며 그것들중 하나는 어떤 다른 주콤퓨터를 지적한다. 자기의 현재등록부에 모든 html파일을 불러 들였다면 문서의 꼬리표들에 있는 FQDN과 경로이름을 제거하지 않고서는 련결부분들을 누르는것으로써 문서를 호출할수 없다. 실례로 2번째 행에서 꼬리표 <A>는 완전한 URL이 아니라 다음과 같이 수정되여야 한다.

```
A HREF="ftpdoc.html"
```

그러면 TRE를 리용하여 우와 같은 파일이름만을 선택해보자. 이것은 생각하는것보다 그리 어렵지는 않다. 우의 항목들을 보면 원천문자렬은 3개의 부분으로 나누어 져 있다.

- \(A HREF="\) - 이것은 첫번째 묶음이며 \1로서 표현된다.
- .*\ / - 이 표현식은 문자 /으로 끝나며 열린 인용부호 "다음에 오는 문자렬을 표현한다. 즉

등록부이름에 해당된다. 여기서 사선기호는 해제시켜야 한다.

- \([^/]*"\) - 이 표현식은 \이 붙은 역사선문자들을 탐색한다. 즉 기본파일이름만을 나타내며 \2로서 표현된다.

그러면 정규식에 이러한 3개의 요소들을 리용한 sed지령을 실행시켜 보자.

```
$ sed 's/\(A HREF="\).*\/\([^/]*"\)/\1\2/' httplinks.html
<LI><A HREF="smail.html">Sendmail</A> The Universal Mail Transport Agent
<LI><A HREF="ftpdoc.html" File Transfer Protocol</A>
<LI><A HREF="news.html">Newsgreoups</A> Usenet News
<LI><A HREF="irc.html">Internet Relay Chat</A> On-Line text conversation
```

지령을 실행시키면 파일이름만이 남는다. 이것은 자주 리용되는 지령렬이다. <IMG>꼬리표 역시 URL들을 참조한다. 그러므로 이에 대해서도 s지령을 리용할수 있으며 훈련 삼아 이것을 해볼수 있다.

TRE는 sed의 가장 기본적인 특징이며 일반적으로 자주 사용된다. 비록 이것이 초기에는 배우기 어렵다고 해도 perl을 배우기 위한 중간단계로서 sed를 리용하는 경우에는 이것을 리해하여야 한다. perl 역시 이러한 수자화된 꼬리표를 리용한다.

sed는 패턴, 공간, 분기기호 ,( ), 표식으로 이루어 진다. 이것을 리용하기는 대단히 어려우며 여기서는 더이상 론하지 않는다. 다시말하여 여기에는 많은 경험이 필요된다.

주해

정규식은 grep와 sed는 물론 려과기능을 가진 awk와 perl, 그리고 편집기 vi와 emacs에 의해서도 리용된다. 이러한 지령들의 GNU판본들은 일부 몇개의 메타문자들을 더 지원하며 POSIX 역시 특별한 패턴을 지적한다. 표현식의 사용법에 대하여 구체적으로 알고 싶다면 부록 3을 참고하시오.

요 약

grep계렬

grep는 어떤 패턴에 대하여 파일을 탐색하기 위하여 리용되는 려과기이다. -c선택항목을 리용하여 패턴에 대하여 선택된 행들의 수를 표시할수 있으며 -l선택항목을 리용하여 파일이름만을 볼수도 있다. 또한 -n선택항목은 행번호를 붙여 준다. -i선택항목은 탐색을 진행할 때 대소문자를 구별하지 않도록 하며 -v선택항목은 패턴을 포함하지 않는 행들만을 선택한다.

awk와 sed와는 달리 grep는 탐색이 실패하면 거짓탈퇴상태를 돌려 준다. 탈퇴상태는 특별한 쉘변수 $?에 보존된다.

egrep는 grep기능의 확장이다. egrep에서는 여러개의 패턴을 구분하기 위하여 문자 |를 리용한다. fgrep는 고정된 문자렬만을 리용하며 다른 두 지령들보다 속도가 더 빠르다.

Linux에서 GNU grep는 비교된 행의 린접행들을 보여 주기 위하여 -A와 -B선택항목들을 제공한다. grep -2는 탐색된 행의 우아래로 각각 2개 행씩 더 보여 준다. 또한 여러개의 패턴을 동시에 탐색할수 있으며(-e) 파일로부터 패턴을 불러 들일수도(-f) 있다.

sed

sed명령은 주소(address)와 동작(action)으로 이루어 져 있다. 매개 행들은 행번호에 의하여 혹은 기호 /으로 닫겨 진 하나 혹은 그이상의 문맥패턴을 지적하는것으로써 주소화된다. 명령은 인용부호에 의하여 닫겨 진다.

sed에는 파일의 임의의 위치에 있는 행들을 표시하기 위한 몇개의 명령들이 있다. 동작 p는 파일의 내용을 출력하며 -n선택항목은 매개 행들이 반복되여 나타나지 않게 한다. !는 선택기준을 반전시킨다.

또한 i, a, c, d지령으로써 행들을 삽입, 추가, 변경, 삭제할수 있다. 그리고 r지령으로 임의의 위치에 있는 파일을 읽을수 있으며 w지령으로 파일의 서로 다른 토막을 개개의 파일로서 보존할수 있다.

sed의 s지령을 리용하면 파일의 어떤 문자렬을 다른것으로 치환할수 있다. 탐색패턴과 치환될 패턴은 grep에서 리용한 정규식일수도 있다.

두개의 사선기호 //는 패턴(기억된 패턴)을 탐색하기 위하여 리용된다. 기호 &는 원천문자렬전체를 목적문자렬에 복사한다.

정규식

grep와 sed에서는 패턴으로서 정규식을 지원한다. egrep와 awk 역시 또 다른 형식의 정규식을 리용한다. 이 두 정규식에서 공통적으로 리용되는 몇개의 메타문자들이 있다. 표 15-2는 grep와 sed에서 리용되는 메타문자묶음이며 표 15-3은 egrep와 awk에 대한것이다.

점(.)은 하나의 문자만을 탐색하며 별표(*)는 그 앞문자에 대하여 하나 혹은 그이상의 문자패턴을 탐색한다(문자가 나타나지 않는 경우도 포함된다).

문자모임은 하나의 문자를 탐색하기 위하여 괄호안에 놓이는 문자들의 묶음이다. 범위지정은 통용기호에서와 비슷하다. 그러나 문자모임에 대한 반전은 문자 ^을 리용한다.

기호 ^와 $은 각각 행의 앞부분, 뒤부분에서 패턴탐색를 리용한다. sed는 이러한 문자들을 해석하기 위한 지령이기때문에 쉘의 간섭을 피하기 위하여 정규식에 인용부호를 붙여야 한다.

+와 ?는(egrep와 awk에 의하여 리용된다.) 별표(*)보다 더 제한된 방식으로 리용된다. ?는 앞문자가 나타나지 않거나 하나만 나타나는 경우를 탐색하며 +는 앞문자에 대하여 하나 혹은 그이상의 출현을 탐색한다. 여러개의 패턴들을 구분문자 |를 리용하여 괄호안에 묶을수 있다.

구간정규식(IRE)은 \{m,n\}와 같이 괄호안에 하나 혹은 한조의 수값을 리용한다. 이 표현식앞에는 항상 하나의 문자가 놓이며 전체적으로 볼 때 이것은 그 문자가 몇번 나타나는가를 지적한다. IRE는 지적된 렬위치에서 혹은 어떤 범위에서 패턴을 찾을 때 아주 유용하다.

꼬리표 붙은 정규식(TRE)은 패턴을 묶기 위하여 \(와 \)을 리용한다. 지령은 패턴을 기억하며 이러한 패턴들에 대하여 꼬리표를 달아 준다. 꼬리표들은 임의의 위치에서 \1, \2 등으로 참조된다. 이 기능을 리용하면 지령은 보다 간단해 진다. 즉 원천문자렬의 일부분을 목적표현식에서 다시 리용하여야 하는 경우에 대단히 편리하다.

시험문제

1. 파일이름이 지적되지 않는 경우에는 emp.lst파일로 생각하시오.
2. 지령 grep a b c를 실행시키면 어떻게 되는가?
3. 파일 foo1, foo2, foo3에서 단어 HTML을 가지고 있는 행들의 총수를 어떤 변수에 보관하시오.
4. 현재등록부에서 문자렬 A HREF를 가지고 있는 파일들을 표시하시오.
5. 파일에서 패턴 <TABLE>을 찾으려면 인용부호를 붙여야 하는가?
6. grep를 리용하여 패턴 <TABLE>을 가지고 있는 행들과 그아래로 5개 행들을 표시할수 있는가?
7. 표현식 gg*은 무엇을 의미하는가?
8. grep를 리용하여 문자렬 harrison과 harris를 어떻게 탐색하겠는가?
9. 표현식 a.*b는 무엇을 탐색하는가?
10. 길이가 15이상인 행들을 표시하기 위한 grep지령의 두가지 방법을 이야기하시오.
11. egrep를 리용하여 패턴 lockwood와 harwood를 탐색하시오.
12. fgrep는 어느 때 리용하는가?
13. 파일로부터 패턴을 얻기 위해서는 egrep와 fgrep를 어떻게 리용해야 하는가?
14. 파일의 모든 행들을 2번 반복하여 표시하시오.
15. 파일의 마지막행을 제외한 모든 행들을 표시하시오.
16. 빈 행을 제거하기 위해서는 어떻게 해야 하는가?
17. 단어 Linux를 Red Hat Linux로 치환하기 위해서는 어떻게 해야 하는가?
18. 패턴이 마지막으로 나타나는 행을 표시하기 위한 지령렬을 쓰시오.

련습문제

1. 통용기호와 정규식의 차이점은 무엇인가?
2. 아래의 지령은 무엇을 하는가? 여기서 $기호들은 무엇을 의미하는가?

```
grep "$SHELL$" /etc/passwd | cut -d: -f1
```

3. 기계에 wc지령이 없는 경우 현재체계를 리용하는 사용자수를 grep지령으로서 어떻게 얻을수 있는가
4. i편집기를 기동하는것과 동시에 문자렬 APPENDIX가 포함되여 있는 마지막행에 유표가 나타나게 하는 지령렬을 쓰시오.
5. grep를 리용하여 파일안의 빈 행을 제거하시오.
6. grep "^*" 은 어떤 처리를 하며 문자 \은 실지로 필요한가?
7. 현재의 등록부에서 쓰기불가능한 파일목록을 보기 위해서는 어떻에 해야 하는가?
8. 점으로 시작하고 점으로 끝나는 행들을 표시하시오.
9. 문자렬 plugin 혹은 plugins를 포함하고 있는 행들을 표시하시오.
10. 다음의 조건에 맞는 행들을 탐색하기 위한 표현식을 작성하시오.

    1) jefferis jeffery jeffreys

```
2) hitchen hitchin hitching
3) Heard herd Hird
4) dix dick dicks dickson dixon
5) Mcgee mcghee magee.
```

11. 파일 emp.lst로부터 생일이 오늘인 사람들의 이름을 보여 주는 grep지령렬을 작성하시오.
12. emp.lst파일에서 director로 되여 있지 않는 가장 젊은 사람들의 이름과 직위를 표시하시오.
13. 지령 grep "**" foo는 어떤 처리를 하는가?
14. 지령 grep *는 어떤 처리를 하는가? 그리고 어떤 때 동작하지 않는가?
15. 아래의 지령을 해석하시오.

```
grep "^[^^]" foo
```

16. 파일에서 bill christie라는 이름을 찾으려고 한다. 그런데 bill이라는 문자렬은 william 혹은 bill 그대로 존재할수도 있고 christie라는 문자렬은 christy라는 문자렬로서 존재할수도 있다. 정확한 표현식을 작성하시오.
17. grep를 리용하여 문자렬 wood, woodcock, woodhouse를 탐색하시오.
18. 이 책에서 단락번호와 보조단락번호를 포함하는 행들을 표시하기 위한 표현식을 작성하시오.
19. romeo의 우편함을 보고 그가 henry로부터 통보문을 받았다는것을 혹은 Subject:행이 단어 urgent 혹은 immediate를 포함하고 있다는것을 그에게 이야기하기 위한 지령렬을 작성하시오.
20. 파일의 시작부분에 <HTML>라는 꼬리표를, 끝에는 </HTML>꼬리표를 추가하시오.
21. 기호 #으로 시작되는 모든 행들을 지우시오(#!/bin/ksh 행만은 제외한다).
22. grep와 sed를 리용하여 aaa, bbb와 같은 련속적이면서도 3개의 동일한 문자로 이루어 진 패턴을 탐색하시오.
23. grep와 sed지령으로써 각각 길이가 100이하인 행들을 찾으시오.
24. 매행의 끝에 있는 공백들을 제거하시오.
25. 아래의 지령에서 무엇이 틀렸는가를 밝히고 그것을 바로 잡으시오.

```
sed 's/compute/calculate/g
s/computer/host/g' foo
```

26. 아래의 지령은 어떤 변화가 있는가?

```
sed 's/print/printf/g
s/printf/print/g' foo
```

27. 파일에서 단어 encryption이 나타나는 수를 표시하기 위한 지령렬을 쓰시오.
28. emp.lst의 date마당에 세기항목을 추가하시오 .
29. HTML파일에서 모든 <B>꼬리표에는 </B>꼬리표가 항상 뒤따른다. 이 꼬리표를 <STRONG>와 </STRONG>으로 바꾸시오.

제 16 장. awk를 리용한 프로그람작성

awk지령은 보고서양식화기능으로서 1977년에 UNIX체계에 편입되였다. awk라는 이름은 Aho, Weinberger, Kernighan이라는 작성자들의 이름의 첫 글자를 따서 만들었다. awk는 UNIX체계에서 가장 강력한 본문처리능력을 가지고 있다(perl은 모든 체계에서 다 쓰이는것이 아니다). sed와 마찬가지로 awk는 여러개의 려과기능들을 결합하였기때문에 보고서서술능력이 대단히 강하다. Linux에서는 이것이 gawk(GNU awk)로 리용된다.

awk는 한가지 처리만을 진행하는 UNIX지령이 아니다. 사실상 그것은 여러가지 처리를 진행하며 성능이 대단히 강하다. 다른 려과기들과 달리 awk는 마당준위에서 처리를 진행하며 개별적인 마당들에 대한 접근, 변환, 양식화 등을 쉽게 할수 있다. 또한 패턴비교를 위한 정규식을 리용하며 C형의 프로그람작성구조, 변수, 내부함수들을 가지고 있다.

여기서는 awk의 중요한 기능들을 상세히 론한다. 이것은 perl을 리해하는데 큰 도움을 줄것이다. 즉 perl은 대부분의 awk구조들을 리용한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- awk의 선택조건과 동작요소들을 가진 문법에 대하여 리해한다(16.1).
- printf로 행을 마당들로 분할하고 출력자료를 양식화한다(16.2, 16.3).
- 조건에 맞는 행들을 선택하기 위하여 비교연산자를 리용한다(16.4).
- 탐색하려는 패턴에 대하여 egrep형의 정규식과 함께 연산자 ~와 !~를 리용한다(16.4).
- 10진수값을 어떻게 처리하며 어떻게 수값계산에 리용하는가를 배운다 (16.4, 16.5).
- 변수를 정의하거나 초기화하지 않고 리용한다(16.6).
- BEGIN, END단락을 리용하여 전처리와 후처리를 진행한다(16.8).
- 쉘스크립트로부터 프로그람을 실행시키고 awk가 스크립트에 제공된 인수들을 읽게 한다(16.9).
- 말단으로부터 입력자료를 얻기 위하여 getline함수를 리용한다(16.10).
- 배렬에 대한 리용법과 비수자적인 첨자를 리용하여 배렬요소에 접근하기 위한 방법을 배운다 (16.12).
- 문자렬처리를 위하여 내부함수들의 리용방법을 배운다(16.13).
- if문의 리용방법에 대하여 배운다(16.14).
- for와 while을 리용하여 고급한 본문조작을 수행한다(16.15).

16.1 awk의 기초

awk는 지령이 아니며 그렇다고 하여 프로그람작성언어도 아니다. awk는 2개의 구성요소로 이루어졌으며 겹인용부호와 괄호를 리용한다.

```
awk 선택항목 '선택조건 {동작}' 파일들
```

이러한 요소들은 find에서와 비슷하다. 선택조건(주소지정형식)은 입력자료를 려과하여 동작요소에 해당한 행들을 선택한다. 동작은 이러한 행들에 대하여 적용되며 이 요소는 괄호 { }에 의하여 닫겨 진다.

주소(선택조건)와 동작은 겹인용부호들에 의하여 닫겨 진 하나의 awk프로그람을 이룬다. 이러한 프로그람들은 여러 행에 놓일수 있지만 대체로 한행으로서 길게 작성한다. 그림 16-1은 대표적인 awk견본프로그람이다.

그림 16-1. awk프로그람의 구성요소

awk에서 선택조건은 sed에서보다 범위가 더 넓다. 즉 그것은 /negropont e/와 같은 어떤 패턴일수도 있고 awk의 변수 NR를 리용한 행주소일수도 있다. 또한 쉘에서 리용되는 연산자 &&, ||(18.6)를 리용한 조건식일수도 있다. 사용자는 실지로 임의의 조건식을 리용하여 행들을 선택할수 있다.

가장 일반적인 awk지령은 주소와 동작으로 이루어 져 있다. 아래의 실례는 우편함으로부터 Subject:행들을 선택한다.

```
$ awk '/^Subject:/ { print }' $HOME/mbox
Subject: RE: History is not bunk
Subject: Mail server problem
Subject: Take our Survey, Win US$500!
```

선택조건(/^Subject:/)부분은 동작({print })부분에 의하여 처리될 행들을 선택한다. 선택조건이 없으면 동작은 모든 행에 대하여 적용된다. 반대로 동작부분이 생략되면 모든 행들을 출력한다. 그러나 둘중의 어느 하나는 반드시 지정되여야 한다.

print문은 마당지적자가 없이 리용될 때 전체 행을 표시한다. 더우기 출력동작은 awk의 기정동작이므로 아래의 세 형식은 본질상 같다.

```
awk '/^Subject:/' mbox                    기정동작을 수행한다
awk '/^Subject:/{ print }' mbox           공백을 허용한다
awk '/^Subject:/ { print $0 }' mbox       $0은 완전한 행이다
```

지금까지의 프로그람들은 파일 emp.lst가 고정길이의 행들로 이루어 져 있기때문에 읽기 쉬운 출력자료를 생성하였다. 이 장에서 awk프로그람들은 입력자료로서 파일 empn.lst를 리용한다. 이 파일에서 모든 행들의 길이는 서로 다르다.

```
$ head -4 empn.lst
2233|charles harris|g.m.|sales|12/12/52| 90000
9876|bill johnson|director|production|03/12/50|130000
5678|robert dylan|d.g.m.|marketing|04/19/43| 85000
2365|john woodcock|director|personnel|05/11/47|120000
```

패턴탐색은 sed형태로서 진행되지만 awk는 egrep형의 정규식(15.6)을 리용한다. 이 형태에서는 패

련구분문자 |와 앞문자에 대한 출현을 제한하기 위한 +와 ?문자를 리용한다.

```
$ awk '/wilco[cx]k?s?|wood(cock|house)/' empn.lst
2365|john woodcock|director|personnel|05/11/47|120000
1265|p.j. woodhouse|manager|sales|09/12/63| 90000
3212|bill wilcocks|d.g.m.|accounts|12/12/55| 850000
2345|james wilcox|g.m.|marketing|03/12/45|110000
```

지금까지는 단 한행의 awk프로그람을 작성하였다. 앞으로 여러개의 행으로 이루어 진 프로그람들을 보게 될것이다. C쉘사용자들은 이와 같은 프로그람을 실행시키려면 [Enter]건을 해제시켜야 한다. 이 문제들에 대하여서는 15.7의 "주해"를 참고하시오.

awk프로그람은 겹인용부호로서 둘러 막아야 한다. 겹인용부호는 정확히 사용되지 않으면 오유를 발생시킨다.

16.2 행을 마당들로 가르기

마당들에 대하여 조작하는 UNIX의 려파기들과는 다르게 awk는 단일구분문자로서 공간들과 타브들의 련속적인 렬(공백)을 리용한다. UNIX의 다른 지령들은 마당구분문자에 대한 해석에서 자유롭지 못하다. cut, paste, sort와 같은 지령들은 분리기호(separator)로서 단일문자를 리용한다.

awk는 공백으로써 혹은 -f선택항목과 함께 지정되는 구분문자(delimiter)로써 마당들을 분할한다. 이 마당들은 그 순서에 따라 변수 $1, $2, $3 등의 순서로 표현된다. 변수 $0은 행전체를 표현한다. 이러한 파라메터들은 겹인용부호안에서 쉘에 의하여 평가되기때문에 awk프로그람들은 겹인용부호를 붙여야 한다. 그러면 이러한 변수들을 어떻게 사용하는가를 보자.

```
$ awk -F"|" '/sales/ {print $2,$3,$4,$6 }' empn.lst
charles harris g.m. sales  90000
gordon lightfoot director sales  140000
p.j. woodhopuse manager sales  90000
jackie wodehouse manager sales  110000
```

awk는 입력자료로부터 마당들을 분할한다. 이전에는 이러한 처리를 cut지령으로 실현하였다. 그러면 이번에는 주소로서 행번호를 반영하는 내부변수 NR를 리용하여 행들을 선택하자.

```
$ awk -F"|" 'NR==3, NR==6 { print NR, $2,$3,$6 }' empn.lst
3 robert dylan d.g.m.  85000
4 john woodcock director  120000
5 barry wood chairman  160000
6 gordon lightfoot director  140000
```

우의 실례에서 awk는 3개의 지령 cut, sed, nl기능들의 결합으로서 볼수 있다. 행주소 NR==3, NR==6은 실지로 sed의 3,6p지령과 같다. NR==3은 어떤 값의 할당이 아니라 하나의 조건식이며 C프로그람작성자들에게 있어서 이것은 처음이 아닐것이다. 연산자 ==는 비교검사를 진행하기 위하여 리용되는 연산자들중의 하나이다.

주해

반점은 일반적으로 마당들이 공백에 의하여 구분되도록 한다. 반점이 없다면 모든 마당들은 련이어 나타날것이다.

16.3 출력의 양식화(printf)

awk의 printf문은 자료를 양식화하여 출력한다. awk는 C언어의 printf함수의 대부분의 양식화기능을 다 가지고 있다. 대표적인 양식들은 다음과 같다.

%s---문자렬

%d---옹근수

%f ---류동소수점수

사용자는 보기 편리한 양식으로 자료를 출력하는데 이러한 출력양식과 함께 표현식을 리용할수 있다.

```
$ awk -F"|" '/true*man/ {
> printf "%3d %-20s %-12s %6d\n",NR,$2,$3,$6 }' empn.lst
13 ronie trueman       executive   75000
15 julie  truman       g.m.        95000
```

실례에서 보는바와 같이 이름과 직위마당은 각각 20, 12문자크기로 출력된다. 여기서 기호 -는 왼쪽으로부터 출력자료를 맞추기 위한것이다. 또한 printf문에서 기호 \n은 행바꾸기를 위하여 리용된다. 이와 같이 awk프로그람에서 다양한 형식들을 사용하다면 출력자료에 대한 조작을 자유롭게 할수 있다.

주해

앞의 실례에서 보여 주는것처럼 내부변수 NR는 모든 행이 선택되지 않는한 행번호를 출력하는데 리용하지 않는것이 좋다. 즉 이 경우에는 다른 변수를 정의하고 사용하시오.

표준출력의 방향절환

문자 >와 |을 리용하여 print와 printf문에 의하여 출력된 자료를 하나의 파일로서 만들수 있으며 또 그에 대하여 련속적으로 지령을 처리할수 있다. 실례로서 아래의 명령문은 printf문의 출력자료를 정렬(sort)시킨다. 이때 지령에 대하여 겹인용부호를 붙여야 한다.

```
printf "%s %-10s %-12s %-8s\n", $1,$3,$4,$6 | "sort"
```

문자 >을 리용하는 경우에도 마찬가지이다.

```
printf "%s %-10s %-12s %-8s\n",$1,$3,$4,$6 > "mslist"
```

즉 awk는 다른 출력흐름에 대한 자료를 조작할수 있는 유연한 기능을 제공한다.

16.4 비교연산자

실례로서 직위가 director이거나 chairman인 사람들에 대하여 3개의 마당만을 표시하려면 어떻게 해야 하겠는가? 이것은 지정된 마당(여기서 $3)과 패턴을 비교하는것으로써 진행할수 있다.

```
$ awk -F"|" '$3=="director" || $3=="chairman" {
> printf "%-20s %-12s %d\n", $2,$3,$6 }' empn.lst
bill johnson       director       130000
```

```
john woodcock      director      120000
barry wood         chairman      160000
gordon lightfoot   director      140000
derryk o'brien     director      125000
```

우에서 보는바와 같이 마당과 패턴의 비교는 이번이 처음이다. 사실상 이러한 비교는 awk와 perl에서만 가능하다. 우의 조건을 반전시키기 위해서는 연산자 != 와 &&을 리용해야 한다.

```
$3 != "director" && $3 != "chairman"
```

즉 우의 프로그람은 자료기지에서 director, chairman들을 제외한 나머지사람들에 대해서만 표시한다. 이러한 비교는 공백마당에 매몰된 문자렬이 아니라 마당전체에 대하여 진행된다. 그러면 파일 emp.lst에 대하여 아래의 프로그람은 어떻게 동작하겠는가?

```
$ awk -F"|" '$3 == "director" || $3 == "chairman"' emp.lst
$_
```

이 파일의 3번째 마당에는 문자렬과 함께 공백들이 매몰되여 있기때문에 마당비교는 다음에 보여 주는 정규식으로 처리한다.

16.4.1 정규식연산자 (~, !~)

패턴을 정규식과 어떻게 비교하겠는가? 이전에는 아래와 같은 방식으로 awk를 리용하였다.

```
awk '/wilco[cx]k*s*|wood(cock|house)/' empn.lst
```

이것은 지적된 마당에서가 아니라 행에 대하여 임의의 위치에 있는 여러개의 패턴들을 탐색한다. 마당에 대하여 탐색을 제한하기 위해서는 연산자 ~(혹은 !~)을 리용해야 한다(!~연산자는 비교를 반전시킨다). 이 연산자를 리용하면 탐색을 더 구체적으로 진행할수 있다.

```
$2 ~ /wilco[cx]k*s*|wood(cock|house)/            두번째 마당을 접합한다
$2 ~ /wilco[cx]k*s*/ && $2 ~ /wood(cock|house)/          우와 같다
$3 !~ /director|chairman/            director 혹은 chairman이 아닌 사람들
```

연산자 ~와 !~는 $1과 $2와 같은 마당지정자들과 함께 리용된다. 우의 프로그람에서 구분문자 |은 egrep에서와 같다. 실지로 awk는 egrep에 의하여 사용되는 모든 정규식들을 리용한다. 그러나 grep와 sed에 의하여 사용되는 IRE와 TRE는 리용할수 없다(15.12).

참고

마당에 매몰된 문자렬을 비교하기 위해서는 ==연산자대신에 ~를 리용해야 한다. 마찬가지로 != 대신 !~를 리용해야 한다.

실례로 직위가 g.m.인 사람들을 탐색하기 위해서는 다음과 같이 작성할수 있다.

```
awk -F"|" '$3 ~ /g\.m\./ { printf "……
```

우의 실례는 문자렬 d.g.m에 해당한 행들도 표시한다. g.m.에 해당한 행만을 탐색하려면 문자 ^와 $를 리용해야 한다. 이러한 문자들은 awk에서는 약간의 다른 의미를 가진다. 즉 여기서는 마당의 시작과 끝을 지적하기 위하여 리용된다. 이것을 고려하여 다시 작성하면 다음과 같다.

```
awk -F"|" '$3 ~ /^g\.m\./ { printf "····
```

여기서 기호 ^은 3번째 마당의 시작부분에서 비교를 진행하도록 한다.

참고

마당의 시작부분에서 문자렬을 탐색하기 위해서는 패턴앞에 문자 ^을 붙인다. 마찬가지로 마당의 끝에서 탐색을 진행하려면 문자 $을 리용해야 한다.

16.4.2 수값비교

awk는 옹근수형과 류동소수점형의 수값을 관리할수 있다. 즉 그러한 수값들에 대한 관계검사를 진행할수 있다. 표 16-1에서 보여 주는 연산자들을 리용하면 로임이 120,000딸라이상인 사람들에 대한 로임명세서를 표시할수 있다.

표 16-1. awk에서 리용되는 연산자

연산자	의 미
<	작다
<=	작거나 같다
==	같다
!=	같지 않다
>=	크거나 같다
>	크다
~	정규식을 정합한다
!~	정규식을 정합하지 않는다

```
$ awk -F"|" '$6>120000 {
> printf "%-20s %-12s %d\n", $2,$3,$6 }' empn.lst
bill johnson        director     130000
barry wood          chairman     160000
gordon lightfoot    director     140000
derryk o'brien      director     125000
```

또한 1945년에 태여났거나 100000딸라이상의 로임을 받는 사람들을 탐색하기 위해서 다음과 같이 표현식들을 결합할수 있다.

```
awk -F"|" '$6 > 100000 || $5 ~/45$/' empn.lst
```

문맥주소 /45$/은 마당의 끝에서 문자렬 45를 비교한다는것을 말해 준다. 이러한 모든 연산자를 리용하여 사용자는 간단한 문자렬이나 정규식으로써 혹은 수값비교식을 작성하는것으로써 행들을 선택할수 있다.

16.5 수값처리

awk는 산수연산자 +, -, *, /, %를 리용하여 수값연산을 진행할수 있다. awk는 쉘에서 리용할수 없었던 류동소수점형의 수값을 관리한다.

사람들은 일상적으로 자기 로임을 제외한 리익금을 얻으려고 한다. 그러면 매 사람들의 1년간 리익

금의 총액이 한달로임과 같다고 가정해 보자. 사람들의 로임명세는 다음과 같다.

```
$ awk -F"|" '$4 == "sales" {
>printf "%-20s %-12s %6d %8.2f\n", $2,$3,$6,$6/12 }' empn.lst
charles harris       g.m.        12/12/52  90000
gordon lightfoot     director    09/03/38  140000
p.j. woodhouse       manager     09/12/63  90000
jackie wodehouse     manager     05/01/59  110000
```

마지막렬은 리익금을 보여 준다. 즉 로임값을 12로 나누는것으로써 얻을수 있다. 또한 실례에서는 류동소수점형의 수값을 표시하기 위하여 %f형식을 리용하였다.

16.6 변수

awk는 NR 그리고 $0과 같은 내부변수를 가지고 있으며 사용자는 자기의 요구에 맞게 변수들을 리용할수 있다. awk에 의하여 리용되는 사용자정의변수는 두가지 특징을 가진다.

- 형선언이 필요 없다.
- 기정적으로 매개 변수들은 형에 따라 0 혹은 NULL문자로 초기화된다. awk는 변수의 형을 식별하기 위한 방법을 가지고 있다.

이러한 특징들을 리용하여 120,000딸라이상의 로임을 받는 사장들에 대하여 유일번호를 붙일수 있다.

```
$ awk -F"|" '$3 == "director" && $6 > 120000{
> kount = kount + 1
> printf "%3d %-20s %-12s %d\n", kount,$2,$3,$6 }' empn.lst
1  bill johnson      director     130000
2  gordon lightfoot  director     140000
3  derryk o'brien    director     125000
```

kount의 초기값은 0 이다. 그것은 첫행에서 그 변수가 수값 1로서 할당되기때문이다. 사용자는 awk 프로그람에서 C형의 증가연산자 ++를 리용할수 있다.

```
kount++                    kount=kount+1과 같다
kount += 2                 kount=kount+2와 같다
printf "%3d\n", ++kount    현시하기전에 kount를 증가시킨다
```

주해

awk프로그람에서 사용되는 사용자정의변수에 대해서는 형선언이나 초기값설정이 필요 없다. awk는 변수들의 형을 식별하여 0 혹은 NULL문자로 자동적으로 초기화한다.

16.7 파일로부터 프로그람의 읽기(-f)

awk프로그람에서 코드량이 많은 경우에는 그 프로그람을 파일로서 보존하여 호출할수 있으며 그러한 파일들에는 .awk라는 확장자가 붙는다. 그러면 먼저 이전에 작성한 프로그람을 파일 empawk.awk로서 보존하자.

```
$ cat empawk.awk
```

```
$3 == "director" && $6 > 120000 {
printf "%3d %-20s %-12s %d\n", ++kount,$2,$3,$6 }
```

이러한 상태에서 awk로 이전과 같은 출력내용을 얻기 위해서는 -f선택항목을 리용해야 한다.

```
awk -F"|" -f empawk.awk empn.lst
```

주의

-f선택항목으로서 awk를 리용한다면 파일에 보존된 프로그람에 인용부호를 붙이지 말아야 한다. awk프로그람이 지령행에서 지적될 때에만 인용부호를 리용한다.

16.8 BEGIN과 END

BEGIN단락은 입력자료를 처리하기전에 어떤 내용을 출력하기 위하여 리용되며 END단락은 처리가 끝난후에 총계를 표시하기 위하여 리용된다. 두 단락들은 아래와 같은 형식을 가진다.

```
BEGIN { 동작 }                둘 다 대괄호를 요구한다
END { 동작 }
```

이러한 단락들은 awk프로그람의 본체에 의하여 구분된다. 우와 같은 단락을 리용하여 시작부분에 적당한 서론을 표시하고 또 처리끝에 평균로임을 표시하기 위한 조작을 할수 있다. empawk2.awk파일로서 아래의 awk프로그람을 보존하시오.

```
$ cat empawk2.awk
BEGIN { printf"\n\t\tEmployee abstract\n\n"
}
$6 > 120000 {  # Increment variables for serial number and pay
            kount++; tot+= $6
            printf "%3d %-20s %-12s %d\n", kount,$2,$3,$6
}
END { printf "\n\tThe averager salary is %6d\n", tot/kount
}
```

awk는 설명문에 대하여 문자 #를 리용한다. 여기서 BEGIN단락은 적당한 서론을 포함하고 있으며 END단락은 선택된 사람들에 대하여 평균로임을 표시한다. 이 프로그람을 실행시키기 위해서는 -f선택항목을 리용해야 한다.

```
$ awk -F"|" -f empawk2.awk empn.lst


             Employee  abstract


1  bill johnson       director    130000
2  barry wood         chairman    160000
3  gordon lightfoot   director    140000
4  derryk o'brien     director    125000

         The average salary is 138750
```

괄호 {는 단락이 시작되는 행에 놓여 있어야 한다. 그러지 않으면 오유통보문이 발생한다.

16.9 위치파라메터

앞의 프로그람은 수값 120,000이라는 수값대신에 변수를 리용하였다면 더 일반적인 형식을 가졌을것이다. awk는 **위치파라메터**(positional parameter)라고 부르는 특별한 변수들을 리용한다. 이 파라메터들은 $1, $2 등으로 표현되지만 마당식별자에 따라 다른 내용을 가진다. awk프로그람을 포함하고 있는 쉘스크립트에 인수를 넘겨 주면 이 인수들은 위치파라메터들에 보존된다.

쉘은 스크립트의 인수들을 표현하기 위하여 동일한 파라메터들을 리용하기때문에(18.4) awk에서 위치파라메터들은 겹인용부호안에 놓여야 한다. 이것은 위치파라메터와 마당식별자를 구분할수 있게 한다.

위치파라메터를 리용하기 위하여서는 awk지령(프로그람이 아니다.)이 아래와 같이 간단히 수정된 다음 쉘스크립트(empabs.sh)에 보존되여야 한다.

```
$6 > '$1'                $6 > 120000을 대신한다
```

매개 쉘스크립트는 실행될수 있는가 하는 실행허가를 요구할것이다(8.2에서 주어 진 명령들에 따른다). 그러면 하나의 인수로서 쉘스크립트를 호출하시오. 이 인수는 awk프로그람에서 리용된다.

```
empabs.sh 1000000
```

이것은 조건에 맞는 행들을 선택하기 위한 자료기지질문식을 작성하는것과 비슷하다. 실례로서 이러한 기능을 리용하여 매 사람들에 대한 평균로임을 계산할수 있다. 하지만 grep와 sed로써는 이러한 처리를 진행할수 없다.

16.10 대화식의 프로그람작성(getline)

모든 프로그람작성언어들은 프로그람실행을 중지하고 말단으로부터 입력자료를 얻기 위한 명령문을 리용한다. 이러한 기능을 awk에서는 getline명령문으로 처리한다. 이 명령문은 다양한 형식의 문법을 가지고 있다. 대표적실례로서 다음과 같다.

```
getline cop < "</dev/tty"              cop에 할당되는 표준입력자료
```

사용자는 BEGIN단락에 우에서 지적된 장치이름 /dev/tty를 리용한 getline명령문을 삽입하는것으로써 처리의 효과를 더 높일수 있다. 앞의 실례를 고쳐 쓰면 다음과 같다.

```
$ cat empawk3.awk
BEGIN { printf "Enter the cut-off salary : "
        getline cop < "/dev/tty"
        printf "\n\t\tEmployee abstract\n\n"
}
$6 > cop { printf "%3d %-20s %-12s %d\n", ++kount,$2,$3,$6
}
```

awk프로그람에서는 파일이름에 대하여 "dev/tty"와 같이 겹인용부호를 붙여야 한다. 우와 같은 스크립트를 실행하면 awk는 실행을 중지하고 값을 입력할것을 요구한다.

```
$ awk -F"|" -f empawk3.awk empn.lst
Enter the cut-off salary : 130000                대화기능

            Employee abstract

 1  barry wood          chairman     160000
 2  gordon lightfoot    director     140000
```

이 스크립트는 사용자로부터 자료를 얻기 위하여 대기상태에 있게 된다. 즉 이러한 특징을 리용하면 사용자와 대화식으로 프로그람을 실행시킬수 있다.

16.11 내부변수

awk는 몇개의 내부변수를 가지고 있다(표 16-2). 이러한 변수들은 자동적으로 할당된다. 앞의 실례에서는 행번호를 표시하기 위하여 변수 NR를 리용하였다. 여기서는 그러한 변수들중의 일부를 간단히 설명한다.

표 16-2. awk에서 리용되는 내부변수

변 수	기 능
NR	현재 읽혀 진 행들의 수
FS	입력마당분리기호
OFS	출력마당분리기호
MF	현재의 행에서 마당개수
FO;ENAME	현재의 입력파일
ARGC	지령행에서 인수들의 수
ARGV	인수들의 목록

awk는 기정적인 마당구분문자로서 공백문자렬을 리용한다. 실례자료기지에서는 마당식별문자로서 |를 리용하였다. 그러므로 이 경우에는 FS변수를 이 마당식별문자로서 재정의해야 한다. 이 식별문자가 전반에 걸쳐 리용되였다면 처리를 진행하기전에 프로그람의 본체에서 기정적으로 이 문자를 리용하도록 FS변수를 BEGIN단락에 정의해야 한다.

```
BEGIN { FS="|" }
```

즉 이것은 지령에서 -f선택항목을 리용하는것과 같다.

주해

awk는 기정구분문자로서 하나의 공백 혹은 타브문자가 아니라 그것들의 조합으로 이루어 진 문자렬을 리용한다.

NF변수는 자료기지의 모든 행들에 대한 마당의 수가 서로 다른 경우에 그것을 정돈하기 위하여 리용된다. 자료입력의 실수로 하여 끼여 들어 온 마당의 수가 6이 아닌 모든 행들을 쉽게 찾을수 있다.

```
$ awk 'BEGIN { FS="|" }
>NF !=6 {
```

```
>print "Record No ", NR,"has ", NF, " fields"}' foo
Record No 6 has 4 fields
Record No 17 has 5 fields
```

변수 FILENAME은 현재 처리되는 파일의 이름을 보존하고 있다. grep와 sed에서와 같이 awk는 지령행에서 여러개의 파일이름을 리용할수 있다. 기정적으로 awk는 파일이름을 출력하지 않지만 필요상 그것을 출력하도록 지적할수 있다.

```
'$6 < 4000 { print FILENAME, $0 }'
```

또한 이 변수를 리용하여 파일들에 대한 서로 다른 처리를 진행할수 있다.

16.12 배렬

awk는 1차원배렬을 관리한다. 배렬의 첨수(index)는 일반적으로 임의의 형식일수 있다. 즉 문자렬도 될수 있다. 배렬에 대한 선언은 필요 없다. 배렬은 그것이 리용되는 시점에서 선언된것으로 본다. 또한 배렬은 명백한 초기화조작을 하지 않는 경우에 자동적으로 0으로 초기화된다.

실례로 매 사람들에 대한 로임과 수수료(로임의 20%)의 총합을 보존하기 위하여 배렬을 사용할수 있다. 아래의 프로그람에서는 tot[]배렬을 리용하였다.

```
$ cat empawk4.awk
BEGIN { FS = "|" ; printf "%44s\n", "Salary      Commission" }
/sales|marketing/ {
     commission = $6*0.20
     tot[1] += $6 ; tot[2] += commission
     kount++
}
END { printf "\t     Average   %5d    %5d\n ", tot[1]/kount, tot[2]/kount }
```

여기서 패턴 sales와 marketing은 행전체에 대하여 탐색을 진행한다. 그러나 실지로 이러한 패턴들은 4번째 마당에서만 나타나기때문에 그 마당에 대하여 탐색이 진행되도록 할수 있다. 프로그람을 실행시키면 로임에 대한 두 요소의 평균값이 출력된다.

```
$ awk -f empawk4.awk empn.lst
                          Salary    commission
              Average     105625    21125
```

C프로그람작성자들은 배렬의 한계를 명확히 지정해야 하는 C언어에 비하여 아주 편리하다는것을 알수 있다. 즉 여기서는 형정의, 초기화가 필요 없다.

16.13 함수

awk는 산수연산과 문자렬처리를 진행하기 위한 내부함수들을 가지고 있다(표 16-3). 인수들은 C형식으로 함수에 넘겨 진다. 또한 매개 인수들은 반점으로 구분되며 괄호안에 놓인다. C언어와는 다르게 함수가 파라메터없이 사용될 때 괄호 ( )는 리용되지 않는다.

표 16-3. awk에서 리용되는 내부함수

함 수	의 미
int(x)	옹근수형의 x값을 돌려 준다
squt	x의 2차뿌리값을 돌려 준다
length	행의 길이를 돌려 준다
length(x)	x의 길이를 돌려 준다
substr(stg,m,n)	stg문자렬의 m번 위치로부터 n개 문자로 이루어 진 부분문자렬을 돌려 준다
index(s1,s2)	문자렬 s1에서 문자렬 s2가 포함되여 있는 위치를 돌려 준다
split(stg,arr,ch)	식별문자 ch를 리용하여 문자렬 stg를 배렬 arr로 분할한다. 마당의 수를 돌려 준다
system("cmd")	UNIX지령 cmd를 실행시켜 완료값을 돌려 준다

일부 함수들은 인수의 개수가 서로 다른 여러가지 형식으로 리용되며 특별히 length함수는 이러한 형식의 하나로서 인수없이 사용할수 있다. 이러한 함수들은 여기서 구체적으로 설명되며 동일한 문법을 리용하는 perl에서 리용할수 있다.

length()

length()함수는 인수의 길이를 결정한다. 만일 인수가 없는 경우에는 전체 행을 인수로서 인식한다. 실례로 1024개 문자를 넘는 행들을 찾기 위하여 인수 없는 length함수를 리용할수 있다.

```
awk -F"|" 'length > 1024' empn.lst
```

length함수는 마당에 대하여 적용할수 있다. 아래의 프로그람은 짧은 이름을 가진 사람들을 선택한다.

```
awk -F"|" 'length($2) < 11' empn.lst
```

index()

index(s1,s2)함수는 문자렬 s1에서 문자렬 s2의 위치를 결정한다. 이 함수는 문자렬에 지적된 문자가 들어 있는가를 검사하는데 특별히 유용하다. 실례로 문자렬 abcde에 b라는 문자가 있는가를 검사하기 위하여 이 함수를 리용할수 있다.

```
x = index("abcde", "b")
```

이 실례에서 돌림값은 2이다.

substr()

substr(stg,m,n)함수는 문자렬 stg로부터 부분문자렬을 따낸다. 여기서 m은 부분문자렬의 시작위치를 지정하며 n은 그 길이를 나타낸다. 문자렬값들은 수값연산에 리용될수 있기때문에 이 함수를 리용하여 자료기지에서 1946년과 1951년사이에 태여난 사람들을 쉽게 표시할수 있다.

```
$ awk -F"|" 'substr($5,7,2) > 45 && substr($5,7,2) < 52' empn.lst
9876|bill johnson |direcotr |production|03/12/50|130000
2365|john woodcock|director |personnel |05/11/47|120000
4290|neil o'bryan |executive|production|09/07/50| 65000
3564|ronie tureman|executive|personnel |07/06/47| 75000
```

awk는 문맥으로부터 표현식의 형을 식별하기 위한 방법을 가지고 있다. 즉 awk는 우의 실례에서

보여 주는바와 같이 substr()함수로 date마당을 식별한 다음 수값비교를 위하여 그것을 수값형으로 변환한다.

주해

awk는 수값변수와 문자렬변수에 대한 특별한 구별이 없다. 즉 어떤 수값계산을 위하여 문자렬을 리용할수도 있다.

split()

split(stg,arr,ch)함수는 구분문자 ch로써 문자렬 stg를 분할한 다음 그것을 배렬 arr[]에 보존한다. 아래의 실례는 YYYYMMDD형식으로서 date마당을 변환한다.

```
$ awk -F\| '{split($5,ar,"/") ; printf "19"ar[3]ar[1]ar[2]}' empn.lst
19521212
19500312
19430419
  ....
```

sed로 이러한 처리를 진행할수도 있지만 이 방법은 5번째 마당을 정확히 선택할수 있기때문에 sed를 리용하는것보다 더 좋다. split()함수는 어떤 값을 돌려 주며 이 돌림값에 대해서는 후에 보게 되는 실례에서 론하기로 한다.

system()

이 함수를 리용하면 보고서의 시작부분에 체계날자를 표시하는것과 같은 처리를 간단히 할수 있다. awk로 임의의 UNIX지령을 실행시키기 위해서는 system()함수를 리용해야 한다. 다음과 같은 실례를 보기로 하자.

```
BEGIN { system("tput clear")}              화면을 지운다
         system("date") }              UNIX의 date지령을 실행시킨다
```

이 절에서 론의한 함수들은 매우 다양하게 리용되기때문에 잘 알아야 한다. 이러한 함수들은 perl에서 다시 리용할것이다.

16.14 if문에 의한 흐름조종

awk는 최근의 프로그람작성언어의 특징들을 다 가지고 있다. 즉 조건명령 if와 순환명령문 while, for와 같은 조종명령문들을 가지고 있다. 이러한 모든것들은 조종지령의 성공과 실패에 따라 본체를 실행시킨다. 조종지령은 명령문의 첫행에 지적되는 하나의 조건식이다.

if명령문에 대해서는 제18장에서도 론의되지만 여기서는 그와 다른 형식을 리용한다.

```
if (조건이 성공이면) {                괄호를 리용해야 한다
  statements
} else {                else는 선택적이다
  statements
}
```

C언어에서와 같이 조종흐름구조는 여러개의 지령들로 이루어 진 경우 괄호 {와 }를 요구한다. 더우

기 조종지령은 괄호 ( )로 닫겨 져야 한다.

지금까지 리용된 대부분의 주소들은 if문에서 사용할수 있다. 이전 실례에서는 주소로서 조건식을 리용하여 120,000딸라이상의 로임을 받는 사람들에 대한 행들을 선택하였다.

```
$6 > 120000 {
```

우와 같은 론리식대신에 if문을 리용하여 동작요소안에 조건식을 놓을수 있다.

```
awk -F"|" '{ if ($6 > 120000) printf ····
```

if조건식에는 비교연산자들과 정규식을 탐색하기 위한 특수기호 !와 !~이 리용될수 있다. 론리연산자 ||와 &&을 결합하여 리용하면 awk프로그람작성은 대단히 쉬워 진다. 표현식들을 탐색하기 위한 패턴의 일부를 if문에서 사용되는 형식에 맞게 표 16-4에서 다시 설명한다.

표 16-4.　　　　　awk의 if구조

지 령	의 미
if(x)	x가 NULL이 아닌 경우
if(NR>+3 && NR<+6)	3번째 행부터 6번째 행까지를 의미한다.
if($3=="director" || $3=="chariman"	3번째 마당이 director 혹은 chairman인 사람들을 정합한다.
if($3~/^g.m/)	세번째 마당은 마당의 시작부분에 g.m.을 포함하고 있다.
if($2!~/[hH]o?uston/)	2번째 마당은 정규식을 정합하지 않는다.

else명령문을 리해하기 위하여 로임이 10만딸라보다 작은 경우에는 수수료가 그의 15%, 그렇지 않은 경우에는 10%로 가정하자. if-else명령문은 아래와 같이 리용할수 있다.

```
if ($6 < 100000)
 commission = 0.15*$6
else
 commission = 0.10*$6
```

우의 if명령문은 간단한 구조로서 다시 쓸수 있다.

```
$6 < 100000 ? commission = 0.15*$6 : commission = 0.10*$6
```

이러한 형식은 C나 perl에서 if-else구조의 론리식을 간단히 하기 위하여 리용된다. ?와 : 은 각각 if와 else를 대신한다.

주해

awk에서 리용되는 if명령문에 대하여 endif 혹은 fi는 리용하지 않는다.

16.15 for와 while에 의한 순환

awk는 2개의 순환명령문 for와 while을 지원한다. 이 두 명령은 조종지령이 True값을 돌려 주는 동안 순환본체를 반복실행시킨다. for문은 두가지 형식을 가지고 있다. 아래의 실례는 첫번째 형식을 보여 준다.

```
for ( k=1 ; k<=9 ; k+=2)              BASIC에서는 FOR k=1 TO 9 STEP 2이다
```

이 형태는 3개의 요소들로 이루어 진다. 즉 첫 요소는 k의 값을 초기화하는 부분이며 2번째 요소는 순환할 때마다 비교되는 조건식이다. 마지막 3번째 요소는 k값을 증가시키는 부분이다.

16.15.1 /etc/passwd로부터 전자우편주소의 생성

여기서는 for문과 split()함수 그리고 한행의 조건식을 리용한 실례응용프로그람을 본다. 이 응용프로그람은 /etc/passwd의 5번째 마당 즉 GCOS마당을 리용하여 전자우편주소를 생성한다. 그러면 passwd파일의 몇개의 행들을 먼저 보기로 하자.

```
henry:!:501:100:henry higgins:/home/henry:/bin/ksh
julie:x:508:100:julie andrews:/home/julie:/bin/ksh
steve:x:510:100:steve wozniak:/home/steve:/bin/ksh
```

주소는 henry_higgins@planets.com.형식으로 되여야 한다. 아래의 awk프로그람은 전자우편주소를 생성한다.

```
$ cat email_create.awk
BEGIN { FS = ":" }
{ array_length = split($5,name_arr," ") ;
  fullname = "";
  for (x = 1 ; x <= array_length ; x++) {
       name_arr[x] = x < array_length ? name_arr[x] "_" : name_arr[x];
   fullname = fullname name_arr[x] ;
  }
  printf "%s@planets.com\n", fullname
}
```

이 프로그람은 GCOS마당($5)을 선택한 다음 공백을 구분문자로 하여 배렬 name_arr에 그것을 분할하여 넣는다. split함수는 요소의 수를 돌려 주며 변수 array_length에 이 돌림값을 보존한다. for순환명령은 배렬로부터 매개 요소들을 선택한 다음 문자 _과 서로 결합한다. 이러한 조작은 마지막요소를 제외한 모든 요소에 대하여 진행된다. passward파일과 함께 프로그람을 실행시키면 다음과 같은 결과가 얻어 진다.

```
$ awk -f email_create.awk /etc/passwd
henry_higgins@planets.com
julie_andrews@planets.com
steve_wozniak@planets.com
......
```

주해

변수들의 련결은 공백을 사이로 그것들을 나란히 놓는것으로써 실현한다. x와 y의 값을 련결하기 위해서는 다음과 같이 한다.

```
print x y
```

16.15.2 for문을 리용하여 발생회수를 계수하기

for문의 두번째 형식은 perl을 제외한 임의의 프로그람작성언어에서 리용되지 않는다. 즉 다음과 같은 형식을 가진다.

```
for ( k in 배렬 )
   지령들
```

이 형식은 배렬을 리용한다. 여기서 k는 첨수이다. 첨수는 옹근수형만으로 제한되지 않는다. 즉 문자렬일수도 있다. 실례로 이 방법을 리용하면 직위에 따르는 종업원들의 수를 간단히 표시할수 있다.

```
$ awk -F"|" '{ kount[$3]++ }                sh를 리용하는 경우에는 첫
> END { for ( desig in kount)                두 행의 끝에 \을 추가하시오
> print desig, kount[desig] }' empn.lst
g.m.        4
chairman    1
executive   2
director    4
manager     2
d.g.m       2
```

우의 프로그람은 사람들의 직위에 따라 종업원자료를 묶기 위하여 먼저 자료기지를 분할한 다음 그 수를 계산한다. 배렬 kount[]은 g.m., chairman, executive 등과 같은 수자가 아닌 값을 첨자로서 가진다. for문은 첨자와 그 출현회수를 출력하기 위하여 END단락에서 호출된다.

주해

실례에서와 같은 론리는 cut, sort, uniq지령을 관흐름으로 결합하여 작성할수 있다(9.18).

16.15.3 while에 의한 순환

while순환문은 조종지령이 성공인 동안 순환을 반복한다. 실례로 전자우편주소를 발생시키기 위하여 리용된 for문을 while문으로 쉽게 바꿀수 있다.

```
fullname = "" ; x = 0 ;
while ( x++ <= array_length ) {
       name_arr[x] = x < array_length ? name_arr[x] "-" : name_arr[x] ;
       fullname = fullname name_arr[x] ;
}
```

16.16 결론

sed와 마찬가지로 awk는 모든 UNIX도구들이 가지는 일반적인 원리 즉《한가지를 수행한다.》는 원리를 깨뜨렸다. 비록 이 장에서 려과도구로서 설명되였지만 그것은 프로그람작성언어보다 더 중요하다. awk는 이전에 요구한것보다 더 많은 기능들을 가지고 있다. 즉 기억을 못할 정도로 대단히 많다.

awk는 도구묶음의 중요한 성원으로서 대단히 편리한점들을 가지고 있다. 그것이 출현하기전까지는

다른 언어들에 설명한 정규식들이 없었다. 또한 수자와 함께 문자렬을 혼합하여 리용할수 없었다. 형선언과 초기화를 하지 않는것으로 하여 awk프로그람은 흔히 C프로그람에서 쓰이군 하였다. 어쨌든 awk는 C와 비슷하며 큰 프로그람에서 C로 코드화하기 위한 중간가동환경으로 리용할수 있다. 에리크 레이몬드는 실지로 그러한 변환기인 awk2c를 작성하였다.

perl은 그의 완전한 기능으로 하여 awk를 완전히 압도하였으며 여러 해에 걸쳐 UNIX도구묶음에 마지막으로 추가되였다. perl이 할수 없는것을 할수 있는 그 어떤 UNIX려파기도 존재하지 않는다. 사실상 perl은 대단히 간결하고 속도가 빠르며 모든 의미에서 다른 려파기들보다 더 우월하다. 이 장은 perl을 리해하는데서 큰 도움을 준다. perl에 대해서는 제20장에서 취급한다.

요 약

awk는 많은 려파기들의 기능을 결합하였다. awk는 행에서 개별적인 마당들($1, $2 등)을 관리할수 있다. 행번호를 지적하기 위하여 sed형의 주소와 내부변수를 리용한다.

자료는 printf로써 양식화되여 출력된다. 양식화지정자들은 문자렬(%s), 옹근수(%d), 류동소수점수(%f)를 양식화하기 위하여 리용된다. 모든 printf명령문은 출력자료를 파일로서 만들기 위하여 혹은 그 자료에 대하여 련속적인 지령을 수행하기 위하여 문자 >와 |을 리용한다.

awk는 또한 비교연산자 >, ==, <= 등을 사용한다. 특수연산자 ~와 !~은 어떤 마당들에서 정규식들을 비교하기 위하여 리용된다. ^와 $은 행에 대하여서가 아니라 마당에 대하여 그의 시작과 끝에서 패턴을 탐색하도록 한다.

awk는 +, -, *, /, %와 같은 표준연산자들을 리용하여 수값계산을 진행할수 있다. 또한 쉘에서는 관리할수 없는 류동소수점형의 수값을 리용할수 있다.

변수들에 대해서는 형정의, 초기화가 필요 없다. awk에서 증가연산자 ++를 리용하여 변수의 값을 증가시킬수 있다.

awk는 외부파일로부터 명령들을 불러 들일수 있다(-f). 이 경우에 프로그람은 인용부호로 닫기지 말아야 한다.

BEGIN과 END단락은 전처리, 후처리를 진행하기 위하여 리용된다. 보고서머리부는 BEGIN단락에 의하여 생성되며 수값의 총액을 계산하는 처리는 END단락에서 진행한다. 또한 쉘스크립트로부터 awk지령을 실행시킬수 있으며 스크립트에 인수들을 넘길수 있다. 이러한 인수들은 '$1', '$2' 등으로서 awk에 의하여 인식된다. 파라메터들은 외인용부호를 붙여야 한다.

getline명령문은 건반으로부터 자료를 얻기 위하여 처리를 중지한다. 이 명령문을 리용하면 awk는 호상 대화하는 방식으로서 쉘스크립트처럼 리용될수 있다.

awk의 내부변수들은 마당식별문자(FS), 마당들의 수(NF), 파일이름(FILENAME)을 지적하기 위하여 사용된다.

awk는 또한 1차원배렬을 리용한다. 1차원배렬에서 첨수는 문자렬일수도 있다. awk는 많은 내부함수들을 가지고 있으며 대부분은 문자렬처리를 위하여 사용된다. 이러한 함수들로서 length, substr, index 등이 있다. system함수는 UNIX지령을 실행한다.

if명령문은 프로그람의 흐름을 조종하기 위하여 조종지령의 돌림값을 리용한다. else명령문은 선택적이다. if명령문에서 모든 연산자들을 리용할수 있으며 추가적으로 복잡한 조건식에 대하여 연산자 ||와 &&을 제공한다.

for순환의 첫 형태는 배렬을 리용하며 비수자화된 첨수를 리용하여 실체의 출현회수를 계산할수 있

다. 두번째 형태는 C에서와 같다. while순환문은 조종지령이 True값을 돌려 주는 동안 명령묶음을 반복 실행시킨다.

perl은 awk보다 우월하다.

시험문제

1. print와 print $0사이의 차이점은 무엇인가?
2. 아래의 명령문에서 틀린것은 무엇인가?

```
print "%s %-20s\n", $1,$6 | sort
```

3. 첫행부터 시작하여 파일의 모든 행들을 선택하시오.
4. emp.lst파일로부터 9월과 10월에 태여난 사람들에 대한 행들을 표시하시오.
5. 길이가 100이상, 150이하인 행들을 표시하시오.
6. 현재등록부에 있는 파일들의 크기를 출력하시오.
7. emp.lst파일로부터 평균지불액을 계산하고 그것을 변수에 보관하시오.
8. for순환명령을 리용하여 지령 echo "CUMENT LIST"의 출력내용을 중심에 놓으시오. 페지너비는 55문자로 한다.

련습문제

1. 등록부 /etc/passwd로부터 Korn쉘 혹은 bash쉘을 리용하는 사용자들에 대하여 그 이름과 쉘이름을 표시하시오.
2. 등록부 /etc/passwd로부터 통과암호를 가지고 있지 않는 사용자들을 표시하시오.
3. 모든 체계사용자들을 무시한후 /etc/passwd에서 다음번에 쓰일수 있는 UID를 찾아 내시오.
4. 파일로부터 모든 빈 행들을 어떻게 지우겠는가?
5. 이해 1월 6일 11시에 마지막으로 수정된 홈등록부의 파일들을 표시하시오.
6. 5번 문제에 대한 지령렬을 날자와 시간이 스크립트로 제공되도록 일반화하시오.
7. 체계지령 tar는 경로이름이 100개의 문자를 초과하면 탐색하지 못한다. 그러면 이와 같은 파일의 목록을 생성하려면 find와 awk를 어떻게 리용해야 하겠는가?
8. 등록부 /etc/passwd에서 다음으로 쓰이게 될 UID를 인쇄하시오.
9. 파일 empn.lst를 100000딸라로임을 기준으로 하여 2개의 파일로 분할하시오.
10. 파일 empn.lst에서 마지막이름이 처음에 나타나도록 매 사람들에 대한 이름을 수정하시오.
11. 표준입력장치로부터 넘겨 진 자료 즉 렬번호를 가지고 그 렬에 해당한 내용을 현시하기 위한 awk 지령렬을 작성하시오. 실례로 지령 <ls -l | 프로그람이름 5>는 목록의 5번째 렬만을 출력한다.
12. 스크립트를 리용하여 PID대신에 그 이름을 지적하는것으로써 처리를 중지하시오.
13. 모든 행들에 대하여 마당이 시작되는 위치가 서로 다른 경우(마당들이 정돈되여 있지 않고 무질서하게 놓여 있는 경우) 어떤 행의 마지막마당의 값을 어떻게 출력하겠는가?
14. 8번 시험문제를 while문을 리용하여 작성하시오.
15. 현재체계를 리용하는 사용자들을 표시하고 몇명인가를 계산하시오.

제 17 장. 환경의 전용화

조작체계와의 호상대화는 사용자들에게 있어서 큰 의의를 가진다. 일상적으로 이 대화는 등록부의 변경, 파일들의 보기, 프로그람의 편집과 번역 ,이전 지령의 끊임 없는 반복으로 이루어 진다. 사용자는 이러한것에 대하여 될수록 간단한 방법들을 리용하였을것이다. 비합리적인 환경설정은 UNIX프로그람작성자와 관리자에게 있어서 불편한감을 줄수 있다. 이로 하여 사용자는 자기 전용의 환경설정을 요구할것이다.

이 장에서는 환경과 관계되는 쉘들의 특징들에 중심을 둔다. UNIX는 쉘의 설정들을 조작하는것으로써 고도로 전용화(customize)될수 있다. 또한 지령들의 기정동작을 변경시킬수 있으며 이전 지령의 재호출, 편집, 재실행을 진행할수 있다. 또한 지령에 대하여 지름법(shortcut)을 만들수 있으며 쉘이 지령들과 파일이름들을 자동적으로 완성하게 할수 있다. 사용자는 이러한 환경설정이 자기가 체계에 가입할 때 그리고 쉘스크립트에서 항상 유효하게 쓰이도록 할수 있다. 전용화의 가능성여부는 자기가 리용하는 쉘에 기본적으로 의존한다. 사용자들은 이 장을 읽은후에 자기의 전용쉘을 선택하고 싶을것이다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- Bourn, Korn, bash, C쉘에 대한 환경과 관련되는 일반적인 특징들을 리해한다(17.1).
- 환경변수의 의미를 배운다(17.2, 17.3).
- 별명을 리용하여 지령행을 간소화한다(17.4).
- 리력을 리용하여 이전 지령을 재호출, 편집, 재실행시킨다(17.5).
- Korn쉘과 bash에서 행내편집기능을 가진 vi와 emacs편집기에 대하여 고찰한다(17.6).
- 파일이름과 지령이름의 완성하기기능에 대하여 본다(17.7).
- 인수 noclobber를 리용하여 우연적인 덧쓰기로부터, 인수 ignoreeof를 리용하여 우연적인 탈퇴로부터 파일을 보호하는 방법을 배운다(17.8.1, 17.8.2).
- ~(물결표)를 리용하여 홈등록부를 표현하는 방법을 본다(17.8.3).
- .과 source지령을 리용하여 보조쉘을 기동시키지 않고 쉘스크립트를 실행시키는 방법을 본다(17.9.1).
- 접속상태에서 명령들을 실행시키거나 보조쉘을 기동시키기 위한 시동파일들을 구성하는 방법을 본다(17.9.3, 17.9.6).

17.1 어느 쉘을 선택할것인가

이전 장에서는 대체로 하나의 쉘안에서 지령을 리용하였다. 때때로 일감을 처리하기 위하여 echo와 sed와 같은 지령들과 함께 기호 \을 리용해야 하였다. 이것은 같은 체계에서조차도 서로 다른 방식으로 동작한다는것을 의미한다. 이러한것은 기본적으로는 아래의 원인들로 하여 발생할것이다.

- 사용자들은 실제상 쉘의 내부지령을 리용한다. 모든 쉘들은 자기자체의 내부지령을 가지고 있으

며 어떤 쉘에서의 지령은 일반적으로 다른 쉘에서 동일한 방식으로 동작하지 않는다.

- 일반적인 환경파라메터들에 대한 설정은 개별적인 쉘들에서 서로 다르다.
- 쉘은 사용자와의 대면부라고 말할수 있으며 사용자가 지령을 처리하는것이 아니라 쉘이 지령을 처리한다. 이전 장에서는 일부 경우 [Enter]건을 누르기전에 기호 \을 입력해야 하였다

쉘은 환경을 결정하는 기본매개물이다. 즉 사용자에게 있어서 가장 가까운 준위이다. Korn쉘, bash와 같은 최근의 쉘들은 대단히 풍부한 특징들을 가지고 있으며 고도로 전용화할수 있다. 자기의 가입쉘로서 어느 하나를 선택하면 그것은 다른 쉘보다 먼저 기동할것이다. 모든 사람들마다 자기가 좋아 하는 쉘이 있으며 이전에는 그러한 쉘로서 대체로 C쉘을 택하였다. C쉘은 제일 초기에 나온 쉘로서 많은 기능들을 가지고 있으며 이것은 최근에 나온 쉘들의 기초를 이루고 있다. C쉘이 오늘까지 존재한것은 바로 이러한 원인에서였다.

사용자는 가입쉘을 설정하기 위하여 체계관리자에게 요구할수 있으며 만일 체계가 chsh지령을 지원한다면 직접 할수 있다. 그러면 여기서는 가입쉘로서 Korn을 선택하자.

```
$ chsh
Password:  ***********
Changing the login shell for henry
Enter the new value, or press return for the default
Login Shell [/bin/csh]: /bin/ksh                이 쉘을 사용한다
$ _
```

지령은 /bin/ksh로서 파일 /etc/passwd의 마지막마당을 변경시킨다. 만일 chsh지령을 지원하지 않는다면 수동으로 혹은 usermod지령을 리용하여 변경시켜야 할것이다.

```
henry:x:501:100:henry blofeld:/home/henry:/bin/ksh
```

자기의 환경을 전용화하기전에 우와 같은 변경이 정확히 진행되였는가를 확인하기 위해서는 아래의 지령을 리용하시오.

```
$ echo $SHELL
/bin/ksh
```

이 장에서는 다음과 같은 쉘들에 대한 환경특징들을 보여 준다.

- Bourne쉘(sh): UNIX체계의 첫번째 쉘
- C쉘(csh): 저수준프로그람작성언어이지만 고수준해석기로서 버클리에 의하여 도입되였다.
- Korn쉘(ksh): 그의 ksh93판본은 우의 2개의 쉘과 비교해 볼 때 대단히 우월하다. 체계에 이러한 판본이 없다면 http://www.kornshell.com으로부터 상업적리용을 목적으로 한 공개판본을 불러 들여 그것을 /bin등록부에 놓으시오. 이 장에서 론의되는 Korn쉘기능의 대부분은 초기의 판본 ksh88에서 유효하다. 17.7에서 《참고》는 Korn쉘의 판본을 찾기 위한 기술을 보여 준다.
- bash(Bourne Again shell): Linux체계에서 리용되는 표준쉘이며 기능적으로 Korn과 대등하다. 또한 bash는 UNIX의 변종들에서도 쓰인다.

표 17-1은 이 장에서 취급되는 쉘들에 대한 특징들을 비교적으로 보여 준다. 매개 쉘들에 대해서는

해당한 절에서 취급된다.

표17-1. 각이한 수식들의 특징비교

특 징	sh	csh	ksh	bash
경로이름	/bin/sh	/bin/csh	/bin/ksh	/bin/bash
국부변수 var의 정의	var=value	setenv var=value	var=value	var=value
환경변수정의	우와 갈지만 export var	set var value	export var=value	export var=value
환경변수들의 현시	export	setenv	export	export
별명	___	제공	제공	제공
별명이름정의	___	alias name value	alias name=value	alias name=value
지령리력과 치환	___	제공	제공	제공
이전 지령의 인수추출	___	제공	___	제공
경로이름자르기	___	제공	___	제공
직렬지령편집	___	___	제공	제공
파일이름완성하기	___	제공	제공	제공
cd -로 등록부들사이의 절환	___	___	제공	제공
cd ~user로 홈등록부의 생략	___	제공	제공	제공
보조쉘없이 스크립트를 실행시키기 위한 지령	___	source	.	.or source
가입파일	.profile	.login	.profile	.bash profile, .profile, .bash_login
환경파일	___	.cshrc	ENV에 의하여 결정됨 (흔히 .kshrc)	.bashrc 혹은 BASH_ENV 에 의하여 결정됨
탈퇴파일	___	.logout	___	.bash_logout

주해 매개의 쉘들에 대하여서는 해당한 절에서 취급된다. 그러나 대부분의 개념적인 내용들은 Bourne쉘에 해당한 절은 물론 제기되는 문제들에 대한 소개부분에서 기본적으로 반영된다. 이것은 정보의 중복을 피하기 위해서이다. 그러므로 어떤 쉘을 리용하려고 해도 그렇고 지적된 절만을 읽으려고 해도 이 부분들을 읽어야 한다.

17.2 환경변수

UNIX체계는 많은 쉘변수(8.11)에 의하여 조종된다. 쉘변수들중의 일부는 초기화과정에 설정되며 또 체계사용이 개시된후에도 설정된다. 사용자가 매번 보조쉘을 기동하면 이러한 변수들중의 일부는 어미로부터 보조쉘에 계승된다. 이 절에서는 **환경변수**(environment variable)와 체계변수의 중요성에 대하여 론하며 또한 자기의 편리에 맞게 그것들의 값을 변경하는 방법을 배운다.

환경변수들은 쉘(국부)변수들과 그 범위에 있어서 다르다. 즉 환경변수들은 전역적이다(반출된다). 이것은 그 변수들이 사용자의 전체 환경(쉘스크립트를 실행하는 보조쉘들, 우편지령들, 편집기들)에서 리용된다는것을 의미한다. 쉘에 대한 중요한 환경과 특별한 변수들은 표 17-2에서 보여 준다.

표 17-2. 미리 정의된 체계변수들

sh	csh	ksh	bash	의 미
HOME	home	HOME	HOME	홈등록부 즉 사용자가 가입할 때 놓이는 등록부
PATH	path	PATH	PATH	지령을 찾기 위하여 쉘에 의하여 탐색되는 등록부목록
LOGNAME	user	LOGNAME	USER 혹은 LOGNAME	사용자의 가입이름
MAIL	mail	MAIL	MAIL	사용자의 우편함파일에 대한 절대경로
MAILCHECK	mail	MAILCHECK	MAILCHECK	들어 오는 우편에 대한 우편검사구간
___	history	___	HISTSIZE	기억기에 보존되는 지령의 수
___	savehist	HISTSIZE	HISTFILESIZE	리력파일에 보존되는 지령의 수
___	항상 .history	HISTFILE	HISTFILE	리력파일
TERM	term	TERM	TERM	말단형
___	cwd	PWD	PWD	현재등록부의 절대경로
CDPATH	cdpath	CDPATH	CDPATH	상대경로이름으로서 리용될 때 cd에 의하여 탐색되는 등록부들의 목록
PS1	prompt	PS1	PS1	1차프롬프트문자렬
PS2	항상 ?	PS2	PS2	2차프롬프트문자렬
SHELL	shell	SHELL	SHELL	사용자의 가입쉘과 쉘탈퇴를 가지는 프로그람에 의하여 호출되는 쉘
___	항상 .cshrc	ENV	BASH_ENV	보조쉘을 실행시킬 때 리용되는 환경파일

17.2.1 Bourne, Korn, bash쉘에서 변수의 사용

set문은 모든 단순변수들과 환경변수(1.7)들의 목록을 현시한다. 그러나 env지령(또는 export문)은 환경변수들만을 현시한다. 아래에 보여 주는것은 Bourne쉘에서 환경변수들에 대한 목록이다.

```
$ env
CDPATH=.:..:$HOME
HOME=/home/romeo
LOGNANE=romeo
MAIL=/var/mail/romeo
MAILCHECK=60
PAGER=/usr/bin/more
PATH=/bin:/usr/bin:/usr/dt/bin:/home/romeo/bin:.
PWD=/home/romeo/project5
PS1='$ '
SHELL=/bin/sh
TERM=ansi
....
```

사용자들의 편리를 위하여 환경변수들은 대문자로 정의된다. 우의 출력자료는 변수가 정의되는 방식 혹은 재할당되는 방식을 보여 준다. 아래의 실례는 말단형을 변경시키는 방법을 보여 준다.

```
$ TERM=vt220
$ _
```

그러면 우와 같이 프롬프트에서 변수들이 정의될수 있는가? 그리고 쉘스크립트에서 그것을 유효하게 할수 있겠는가? TERM과 같은 몇개의 변수들에 대해서는 적용된다. 그러나 아래와 같은 변수할당에 대해서는 그렇지 못하다.

```
x=5                x는 국부변수이다
```

여기서 x의 값은 전역적으로 유효하지 못하며 현재의 쉘내에서만 유효하다. 그러나 사용자가 이 변수를 **반출**(export)하면 환경변수로 변환된다.

```
export x           x는 환경변수로 된다
```

Korn과 bash쉘들에서는 한개의 명령문(export x=5)으로 우의 두개의 동작을 결합할수 있다. env지령에 의하여 현시되는 변수들(set로도 볼수 있는 변수들)과 함께 export지령을 쓰지 않는 리유는 그러한 일감이 이미 수행되였기때문이다. 사용자가 가입할 때 체계변수들은 가입쉘에 의하여 실행되는 여러개의 기동스크립트들에 의하여 반출된다. 19.6에서는 export지령을 리용하여 반출된 변수와 그러지 않은 변수들사이의 명백한 차이점을 준다.

참고

사용자들은 체계변수들과의 구별을 쉽게 하기 위하여 자기의 변수들에 대해서는 소문자로 정의하는것이 좋다.

17.2.2 C쉘에서 변수의 사용

C쉘에서 비전역변수 및 전역변수들에 대한 값주기방법은 다른 쉘에서와 좀 다르다. 비전역변수들은 set문에 의하여 값이 할당된다.

```
% set x = 20              =주위의 공백들은 불필요하다
% echo $x
20
```

C쉘은 =주위에 공백을 요구하지 않지만 여기서는 독자들의 편리를 위해 공백을 주었다. 이 x의 값은 전역적으로는 유효하지 못하다. 모든 쉘스크립트에서 유효하게 하자면 setenv문을 리용하여 값을 대입하여야 한다.

```
setenv x 30              =가 없다
```

setenv는 대입연산자 =를 리용하지 않으며(초학자들은 여기서 흔히 오유를 범한다.) 변수와 그의 값들은 나란히 놓인다. 여기서 set와 setenv에 같은 변수이름을 사용하였는데 그러면 x의 값은 얼마이겠는가?

```
% echo $x
20                 국부값
```

echo문은 국부변수에 대한 값을 현시한다. 그러나 보조쉘에서 이와 같은 문을 리용한다면 전역변수에 해당한 값이 현시될것이다.

```
% csh               C의 보조쉘을 만든다
% echo $x
30                 전역값
```

국부변수는 그것들이 둘 다 정의된 쉘에서는 전역변수의 값을 무시한다. 그러지만 사용자가 setenv로 전역변수를 정의하면 그 값은 현재의 쉘에서도 유효하다. 이러한 혼돈을 피하기 위해서는 국부변수와 환경변수들의 이름을 서로 다른 이름으로 주는것이 좋다.

set문은 전역적으로 유효한 변수들이 아니라 현재의 쉘에서 유효한 모든 변수들을 표시한다. 아래의 set문에서는 연산자 =를 리용하지 않으며 출력자료에서 변수와 그에 해당한 값들은 공백으로써 구분된다.

```
% set
cwd      /home/julie/docs
history     100
home        /home/julie
mail        /var/mail/julie
path        (/bin /usr/bin /usr/ucb .)
prompt      %
savehist    100
shell       /bin/csh
term        AT386
user        julie
x           20                 국부변수를 보여 준다
```

set문은 x의 값으로서 20을 현시하였다. 일반적으로 사람들은 우와 같은 변수들을 환경변수로서 생각할수 있는데 명백히 이러한것들은 환경변수가 아니다. setenv문에 의하여 설정된 변수들만이 환경변수로 된다. 아래에 보여 주는 변수들은 setenv문을 인수없이 실행시켰을 때 표시되는 목록이다.

```
% setenv
HOME=/home/julie
PATH=/bin:/usr/bin:/usr/ucb:.              다른 형식
LOGNAME=julie
TERM=AT386
SHELL=/bin/csh
MAIL=/var/mail/julie
PWD=/home/julie/docs
USER=julie
OPENWINHOME=/usr/openwin
```

```
x=30                     환경값을 보여 준다
```

다른 쉘들의 set지령들에서도 같은 형식으로 출력된다. setenv는 x의 전역값 30을 표시한다(set문은 국부값 20을 현시한다). 혹시 home, term, path와 같은 많은 소문자로 되여 있는 체계변수들을 대문자로도 본적이 있는가? 이것은 혼란을 조성하는 시초이기때문에 먼저 지적하지 않을수 없다.

vi와 같은 프로그람들이 대문자환경변수들을 리용한다면 C쉘자체는 소문자로 되여 있는 국부변수들을 허용한다. 실례로 C쉘은 지령탐색을 위하여 PATH변수가 아니라 소문자로서 path변수를 리용한다. 대부분의 이러한 국부변수들은 그에 대응하는 환경변수들을 가지고 있다. 즉 echo $TERM과 echo $term은 같은 값을 출력한다. 일부 체계에서는 변수 term과 user를 변경시키면 자동적으로 TERM과 USER변수도 갱신되지만 그 반대현상은 일어 나지 않는다. 만일 환경변수를 그와 대응한 쉘변수의 복사판으로 본다면 그들사이의 관계에서는 어느 정도의 복잡성이 제기되며 이 책에서는 그러한 내용을 취급하지 않는다.

여기에 set형식과 setenv형식의 차이점을 보여 주는 변수가 있다. 이러한 변수가 바로 path변수이다. 이 변수는 괄호 ( )로 둘러 막히고 공백으로 구별되는 등록부들을 보여 준다. 이러한 내용은 후에 취급하기로 한다. 그 리유는 C쉘배렬에 값을 대입하는 방법을 배운후이면 알게 될것이다.

주해

외부프로그람들은 간단한 쉘변수가 아니라 환경변수들을 참조한다. TERM, ATH, SER와 같은 환경변수들중의 일부는 그에 대응하는 쉘변수들로부터 복사된다. 이상할 정도로 C쉘 그자체는 전역적으로 유효하지 않는 소문자변수들을 사용한다.

17.3 환경변수의 의미

set, env, export, setenv문들의 출력에서 볼수 있는 변수들은 체계의 동작을 조종한다. 사용자들은 많은 경우에 이러한 변수들을 보게 되며 UNIX에 대한 경험을 쌓는 과정에 그 변수들의 값을 변경시킬것이다. 여기서는 먼저 Bourne쉘에 대해서 론의한다. 이 쉘에서의 변수들은 Korn이나 bash에서도 리용된다. 또한 그 일부는 C쉘에서도 리용할수 있다. 쉘에 관계없이 이 변수들의 의미를 알자면 다음의 항목을 읽어야 한다.

17.3.1 Bourne, Korn, bash쉘의 환경변수들

지령탐색경로(PATH)

PATH변수는 지령을 실행시킬 때 그 실행코드에 대한 경로이름을 반영하는 중요한 체계변수들중의 하나이다. 그의 현재값을 보기로 하자.

```
$ echo $PATH
/bin:/usr/bin:/usr/dt/bin:/home/romeo/bin:.
```

즉 이 변수는 임의의 실행가능한 지령이 위치하는 경로에 대하여 쉘에 알려 준다. 우에서 두점(:)에 의하여 구분되는 5개의 등록부들을 볼수 있다. 또한 행의 끝에 한개의 점이 있는데 이것을 놓쳐서는 안된다. 즉 점은 현재의 등록부를 나타내며(6.8) 쉘은 앞의 등록부들에서 탐색이 실패한 경우에 현재등록부에서 지령을 탐색하게 된다. 이에 대해서는 2.2에서 이미 론의하였다.

탐색목록에 /usr/xpg4/bin와 같은 등록부를 추가하려면 이 변수를 다음과 같이 재정의해야 한다.

```
PATH=$PATH:/usr/xpg4/bin                등록부의 추가
```

새로운 등록부는 C쉘에 의하여 현재등록부다음으로 탐색되는 등록부이다. 즉 마지막으로 탐색된다. 현재 이 등록부에는 Solaris의 POSIX호환도구들이 들어 있다. 여기서 이 등록부의 grep지령은 /bin(또는 /usr/bin)에서와는 약간 다르게 동작한다.

주해

두개의 서로 다른 등록부들에 같은 이름을 가진 지령이 있다면 PATH목록에서 앞에 놓인 등록부의 지령이 먼저 실행된다. 즉 홈등록부의 cat지령으로서 cat foo를 호출하려 한다면 홈등록부의것이 아니라 /bin등록부의 cat지령이 실행된다. 이것은 홈등록부가 PATH설정에서 bin등록부보다 후에 놓여 있기때문이다.

홈등록부(HOME)

UNIX체계에 가입(login)하면 사용자는 가입이름뒤에 놓인 등록부로부터 작업을 시작한다. 이 등록부를 **홈등록부** 또는 **가입등록부**(login lirectory)라고 하며(6.5) 변수 HOME에 기록된다.

```
$ echo $HOME
/home/romeo
```

사용자의 홈등록부는 파일 /etc/passwd에서 해당한 행에 반영된다. 이 파일의 매개 행들은 개별적인 사용자들에 해당한것이며 7개의 마당으로 이루어 졌다. 아래에 romeo라는 사용자에 대한 행을 보여 준다.

```
romeo:x:208:50::/home/romeo:/bin/sh
```

홈등록부는 마지막으로부터 두번째 마당에 정의된다. 사용자가 체계에 가입하면 가입프로그람은 이 파일을 읽어서(10.4) 해당한 값으로서 변수 HOME과 SHELL을 설정한다. 파일 /etc/passwd은 체계관리자에 의해서만이 수동적으로 혹은 useradd나 usermod지령들을 써서 편집된다. 그러므로 HOME에 설정되는 값을 변화시키자면 관리자에게 의뢰해야 한다.

사용자는 HOME변수의 값을 변경시킬수 있다. 그러나 이것은 홈등록부를 변경시키는것이 아니라 cd지령이 인수없이 사용될 때 이행하는 등록부만을 변경시킨다. 이것은 인수없이 쓰인 cd지령이 기정적으로 cd $HOME으로 되기때문이다.

참고

만일 자기가 어떤 사람의 .xinit.rc(12.13)을 사용하고 싶다면 HOME의 값을 수정한 다음에 X를 기동시켜야 한다.

```
HOME=/home/julie xinit                같은 행에서
```

이 경우 julie의 .xinit.rc를 자기의 등록부로 복사할 필요가 없다. X는 자동적으로 이 파일을 읽는다. 우에서는 지령호출에 앞서 HOME변수에 먼저 값을 대입하였다. 이 경우 그것들사이에 지령구분문자로서 반두점 ;을 리용하지 않으면 변수의 값은 림시적으로 변경된다. 즉 프로그람의 실행이 완료되면 본래값으로 복귀된다.

사용자이름(LOGNAME)

이 변수는 사용자이름을 보여 준다. 사용자가 파일체계내에서 여기저기 이동하면 자기의 가입이름을 잊어 버리는 경우도 있다(이런 일은 흔치는 않지만 사용자가 여러개의 등록자리를 가지고 있을 때 주로 발생한다). 따라서 때때로 다음과 같은 방법으로 자기의 가입등록자리를 확인해야 한다.

```
$ echo $LOGNAME
romeo
```

다른 곳에서 이 변수를 사용하여 보았는가? 실지 스크립트를 호출하는 사용자에 따라서 서로 다른것을 수행하는 쉘스크립트에서 이 변수를 사용할수 있는것이다.

man에 의하여 사용되는 페지화프로그람(PAGER)

많은 체계들에서 man지령은 페지화된 출력자료를 산출하는데 리용되는 프로그람을 결정하기 위하여 PAGER변수의 값을 사용한다. 최근의 UNIX체계들에서는 이러한 페지화프로그람으로서 more를 리용한다. 그러나 이전 체계들에서는 이와 같은 프로그람으로서 pg를 리용하였다. 만일 체계가 more를 지원하는데도 불구하고 PAGER변수의 값이 pg로 설정되여 있다면 즉시 PAGER의 값을 변경시켜야 한다.

```
$ PAGER=/usr/bin/more ; echo $PAGER
/usr/bin/more
```

만일 체계가 less를 지원하는 경우에는(Linux체계의 man에서 기정적으로 리용된다.) more대신에 less를 사용해야 한다. less는 모든 측면에서 볼 때 more(pg가 아니라)보다 더 우월하며 사용하기도 아주 쉽다.

주해

일부 UNIX체계들에서는 이 변수를 쓰지 않고 파일 /etc/default/man에 페지화프로그람을 정의하고 있다. 만일 사용자가 자기 체계에 이 파일을 가지고 있다면 거기서 아래와 같은 항목을 볼수 있을것이다.

```
PAGER = /usr/bin/more
```

Solaris체계는 PAGER가 정의되여 있지 않으면 more -s를 사용한다.

전자우편함의 위치와 우편검사(MAIL, MAILCHECK)

UNIX의 우편처리체계는 우편이 도착하였다는것을 사용자에게 통지하지 않는다. 즉 이 일감은 쉘에 의하여 처리되여야 한다. 쉘은 변수 MAIL로부터 전자우편함의 위치를 알아 낸다. 이 우편함의 위치로서 이전 체계들에서는 보통 /var/mail, /var/spool/mail, /usr/spool/mail을 리용하였다. 즉 romeo의 우편은 SVR4체계에서는 /var/mail/romeo에 보존된다.

변수 MAILCHECK는 쉘이 새로운 우편이 도착할 때 파일을 검사하는 방법을 결정한다(큰 체계들에서는 대체로 600초이다). 쉘은 마지막검사이후에 변경된 파일을 발견하면 다음의 통보문을 사용자에게 통지한다.

```
You have mail in /var/mail/romeo
```

이때 romeo가 어떤 지령을 실행하고 있다면 그것이 완료된후에 마지막으로 이 통보문을 받게 된다.

프롬프트문자렬(PS1, PS2)

쉘은 PS1과 PS2에 두개의 프롬프트문자렬을 가지고 있다. 첫 프롬프트문자렬 PS1은 흔히 보던것이다. 만일 사용자가 지령을 다음행에 련결하여 계속 쓰려고 하는 경우에 체계는 프롬프트문자 >를 내보낸다.

```
$ sed 's/STRONG/BOLD/g
> s/html/HTML/g'
```

프롬프트문자 >는 PS2에 보존되는 두번째 프롬프트이다. Bourne쉘에서는 보통 PS1과 PS2를 각각 문자 $과 >로 설정한다.

사용자는 Windows환경이 더 좋다고 생각되면 첫번째 프롬프트문자를 C>으로 바꿀수 있다.

```
$ PS1="C> "
C> _
```

$는 가장 일반적으로 리용되는 프롬프트문자이지만 체계관리자는 프롬프트문자로서 #를 사용한다. PS1에 대해서는 Korn과 bash쉘들을 설명할 때 좀 더 자세히 론의한다.

등록부탐색경로(CDPATH)

사용자는 파일체계내에서 다른 등록부에로 이행하여 갈수 있다. 실례로 홈등록부아래에 두개의 등록부 bar1과 bar2가 있고 현재사용자가 등록부 bar1에 위치하고 있다고 하자. 이 경우에 bar2로 옮겨 가야 한다면 보통 cd ../ bar2이라고 쓸수 있다. 즉 사용자는 먼저 변수 CDPATH를 설정하는것으로써 등록부지정을 생략할수 있다.

```
CDPATH:.:..:/home/romeo/project5
```

이것은 쉘이 등록부를 찾을 때마다(그 순서대로) 탐색하는 3개의 등록부들에 대한 문자렬이다. bar1 등록부에서 지령 cd bar2를 쓰면 쉘은 먼저 bar2를 현재등록부(.)에서 찾으며 실패하면 어미등록부(..)를 탐색한다. bar1과 bar2는 둘 다 같은 준위에 있으므로 쉘은 여기에서 bar2를 찾게 된다.

```
$ pwd
/home/romeo/bar1
$ cd bar2 ; pwd
/home/romeo/bar2
```

쉘은 /home/romeo에서 bar2를 찾지 못한 경우에는 다음으로 등록부 /home/romeo/project5에서 찾아 본다.

쉘에로의 탈퇴지령들에 의하여 리용되는 쉘(SHELL)

변수 SHELL은 사용자가 현재 리용하고 있는 쉘에 대하여 알려 준다. 사용자는 vi , emacs , telnet와 같은 프로그람들이 쉘에로의 탈퇴경로를 어떻게 가지는가를 본적이 있을것이다. SHELL은 이 프로그람들이 사용하는 쉘을 결정한다. vi가 쉘에로 탈퇴하기 위해서 :sh지령을 리용한다 해도 실지 호출되는 쉘은 변수 SHELL에 의하여 결정되며 반드시 Bourne쉘로 되는것은 아니다.

HOME과 비슷하게 관리자는 사용자등록자리를 만들 때 파일 /etc/passwd에 가입쉘을 설정한다. HOME변수를 론의할 때 실례에서 본 파일 etc/passwd의 마지막마당은 변수 SHELL의 값을 반영한다. chsh지령은 이 마당을 변경시킨다.

말단형(TERM)

변수 TERM은 현재 사용되는 말단의 형태를 지정한다. 모든 말단은 등록부 /usr/lib/terminfo (Solaris에서는 /usr/share/lib/terminfo)내의 개별적인 조종파일에 정의되여 있는 정확한 특징들을 가지고 있다. 이 등록부는 영문자로 시작되는 이름을 가진 몇개의 보조등록부들을 가지고 있다. 말단의 조종파일은 그 말단이름의 첫 문자를 자기 이름으로 가지고 있는 등록부에 있다. 실례로 ansi말단들은 파일 /usr/lib/terminfo/a/ansi를 사용한다.

vi편집기와 같은 도구들은 말단에 의존하므로 이러한 도구들은 사용자가 리용하고 있는 말단의 형태를 알고 있어야 한다. TERM이 정확히 설정되지 않으면 vi는 동작하지 않는다. 변수 TERM은 원격콤퓨터에 가입할 때도 중요하다. 흔히 많은 UNIX도구들은 TERM이 정확히 설정되지 않은것으로 하여 동작하지 않거나 화면에 이상한것을 표시한다.

17.3.2 C쉘의 특수한 변수들

C쉘이 환경변수로서 대문자변수이름들을 사용한다 하여도 사용자들은 소문자로 된 C쉘변수들의 값을 관리하게 된다. 사실상 이러한 변수들중의 일부는 자기와 대응되는 대문자변수를 가지지 못한다. 그러나 이 변수들은 set명령문으로서 할당된다.

1차프롬프트(prompt)

C쉘은 prompt변수에 프롬프트문자렬을 보관하고 있다. 프롬프트문자렬로서 %를 많이 볼수 있는데 그것을 바꾸자면 set를 사용할수 있다.

```
% set prompt = "[C>] "
[C>] _
```

2차프롬프트문자렬은 ?인데 그 값은 그 어떤 환경변수에도 기억되지 않는다. 더우기 prompt변수는 자기와 대응되는 대문자변수가 없다(그러나 보조쉘들에서는 유효하다).

프롬프트에서 사건번호의 리용(!)

C쉘에는 이전의 지령들을 다시 넣지 않고도 실행시킬수 있게 하는 리력기능이 있다. 이러한 리력기능에 대해서는 후에 보기로 한다. 리력목록에서 매 지령은 자기에게 할당된 사건번호를 가지고 있으며 이 목록에서 바로 그 사건번호를 취하여 지령을 호출할수 있으며 또 실행시킬수 있다.

모든 지령들에 사건번호를 할당하자면 이 속성을 리용한다. 이를 위하여 요구되는 문자는 기호 !인데 이 기호는 \에 의하여 의미해제(escape)되여야 한다.

```
% set prompt = '[\!] '
[12] _                  set prompt는 목록에서 11번째 지령이다
```

기호 !는 이 쉘에서 특수한 의미를 가지므로 기호 \에 의하여 숨겨 지게 된다. 이제부터 사용자가 지령을 입력하면 리력목록에 그 지령이 추가되고 사건번호가 증가된다. 여기서는 개별적으로 이러한 지령리력에 대하여 취급한다.

주해 prompt의 정의에서 기호 !뒤에 공백이 아닌 문자]가 오기때문에 !의 의미를 해제시키는데 \ 을 사용하였다. C쉘은 기호 !뒤의 임의의것을 지령으로 해석하며 문자렬로 시작되는 마지막지령을 반복한다.

지령탐색경로(path)

이것은 Bourne의 환경변수 PATH를 표현하기 위한 C쉘의 방법이다. path목록은 좀 달리 표시된다.

```
% echo $path
/bin /usr/bin /usr/lib/java/bin /usr/dt/bin
```

Bourne에서와 달리 공백으로 구분되는 등록부들의 목록을 볼수 있다. 좀 더 깊이 있게 본다면 set

지령의 출력자료로서 괄호( )로 둘러 막힌 부분이다.

```
path        (/bin /usr/bin /usr/lib/java/bin /usr/dt/bin)
```

이것은 실제상 4개 요소로 이루어 진 배렬이다. 배렬조종에 대해서는 부록 1에서 구체적으로 취급한다. 우리가 알아야 할것은 이 변수들이 재할당되는 방법이다. path목록에 /usr/xpg4/bin을 추가하자면 다음과 같이 해야 한다.

```
% set path = ($path /usr/xpg4/bin)
% echo $path
/bin /usr/bin /usr/lib/java/bin /usr/dt/bin /usr/xpg4/bin
```

setenv지령은 등록부들에 대한 같은 목록으로서 변수 PATH를 Bourne쉘의 형식(17.2)으로 보여준다. path를 바꾸면 PATH도 갱신되며 그 반대도 가능하다.

전자우편함들과 간격검사(mail)

C쉘에서는 Bourne쉘에서의 변수 MAIL과 MAILCHECK의 기능을 결합하여 변수 mail에 기록한다. 17.2에서는 set문이 mail의 값으로서 한개의 파일이름만을 보여 주었지만 이 변수는 여러개의 파일이름들로 설정될수 있으며 선택적으로 한개의 수자가 앞에 붙을수 있다.

```
set mail = (600 /var/mail/julie /opt/Mail/julie)
```

쉘은 매 600초마다 우편이 새로 도착하였는가를 이 두 파일의 마지막변경시간으로써 검사한다. 환경변수 MAIL은 단순히 일부 외부프로그람들에 의하여 읽혀 지는 하나의 파일이름을 보존하고 있다.

그밖의 변수들

아래에 몇개의 다른 변수들에 대하여 간단히 설명한다.

- cwd: 현재등록부를 보존한다.
- user: 가입한 사용자의 이름(Bourne에서의 LOGNAME)
- home, cdpath, shell, term: 이 변수들은 Bourne쉘에서 대문자로 되여 있는 변수들과 같은 의미를 가진다. C쉘도 환경변수로서 SHELL과 TERM을 사용한다.
- history, savehist: 리력기능에 필요한 변수들이면 17.5에서 구체적으로 론의한다.

이 변수들외에도 값이 없는 특수한 변수들이 있는데 그것들중에는 설정된것도, 비설정된것도 있다. 이것들은 《set 변수》형태로 주어 지는데 그것들중 필요한 4개 변수만을 아래에 설명한다.

- notify: 배경일감이 완료되면 사용자들에게 통지한다.
- filec: 파일이름완성에 리용되며 17.7의 C쉘에 대한 설명에서 간단히 론의된다.
- noclobber, ignoreof: 17.8에서 론의된다.

지금까지 C쉘변수들의 세가지 형태에 대해 보았다. 첫째로 prompt변수와 같이 하나의 값으로 설정되는 변수들이며 둘째로 path처럼 여러개의 값을 가진 배렬이며 셋째로 noclobber와 같이 값을 가지지 않는다.

17.3.3 Korn쉘의 기타 환경변수들

Korn쉘은 Bourne의 상위모임이다. 우리가 앞에서 론의한 Bourne쉘의 변수들은 Korn쉘에 대해서

도 물론 적용된다. 그러나 Korn쉘도 역시 자기의 고유한 변수들을 가지고 있다.

프롬프트내에서 현재등록부(PWD와 PS1)

Korn쉘은 변수 PWD를 리용하여 현재등록부의 경로를 기억한다. 이것은 C쉘의 cwd보다 용도가 더 높다. 즉 $PWD를 변수 PS1의 값주기로서 리용할수도 있다.

```
$ PS1 = '[$PWD] '
[/home/romeo] _
```

프롬프트가 $로부터 [/home/romeo]로 어떻게 바뀌는가를 보시오. 그러면 이제 등록부를 cgi로 변경시키자.

```
[/home/romeo] cd cgi
[/home/romeo/cgi] _                    프롬프트는 변경된 등록부를 반영한다
```

PWD는 그리 잘 리용되지 않는 변수로서 작업등록부가 변할 때마다 재평가된다.

그러므로 우에서와 같이 프롬프트는 PWD의 새로운 값을 반영하게 된다. C쉘에서 cwd로서 꼭 같은 조작을 해보면 동작하지 않는다(별명을 사용하면 이와 같은 동작을 실현할수 있다).

PS1에서 사건번호의 사용(!)

Korn쉘은 이전의 지령들을 다시 호출하거나 실행시키는 리력속성을 가지고 있다. 사용자는 PS1프롬프트를 설정하는것으로써 현재의 사건번호를 보여 줄수 있다. 이 특징은 C쉘로부터 유래된것으로서 대입에서 사건번호를 나타내는 기호 !를 리용한다.

```
$ PS1="[!] "                Korn에서는 사선기호 /를 요구하지 않는다
[42] _
```

더 좋기는 여기에 PWD변수를 포함시켜 현재등록부를 반영하게 할수 있다. 이 경우 변수값을 설정할 때 $PWD가 겹인용부호를 요구하지 않으므로 외인용부호를 붙여야 한다.

```
$ PS1='[! $PWD] '                변수는 외인용부호안의것이 평가된다
[43 /home/romeo/project3]
```

지령이 실행될 때마다 사건번호는 증가된다(여기서는 42로부터 43으로 된다). 사건번호에 대한 지식은 이 번호를 참조하여 이전의 지령들을 재실행시킬수 있으므로 아주 쓸모 있다.

그밖의 변수들

Korn쉘에서 사용하는 몇가지 다른 변수들도 있는데 이것들은 set문으로 출력될 때 현시될수도 있고 현시되지 않을수도 있다.

HISTFILE, HISTSIZE는 지령리력기구에서 사용되며 17.5에서 론의하기로 한다.

ENV는 쉘이 호출될 때마다 실행되는 스크립트를 지적한다. 이에 대해서는 17.9에서 론의한다.

EDITOR, VISUAL은 행내편집(in-line editing)기능에 의하여 사용되는(vi혹은 emacs)방식을 정의한다. 이것은 17.6에서 론의한다.

17.3.4 bash의 기타 환경변수들

bash쉘도 Korn쉘처럼 Bourne쉘의 상위모임이며 Bourne쉘에서 설명된 많은 기능들이 bash쉘에도

적용된다. bash도 자기자체의 특징적인 몇가지 변수들을 가지고 있다.

프롬프트내의 현재등록부(PWD와 PS1)

Korn과 비슷하게 bash도 pwd지령을 대신하는 PWD변수를 사용한다. 흔히 그것은 Korn에서와 동일한 방식으로 사용한다.

```
$ PS1='$PWD> '            외인용부호가 리용되여야 한다
/home/juliet> _
```

PWD는 현재등록부가 변경될 때마다 재평가된다. 이것은 사용자가 등록부를 변경시키면 프롬프트도 변경된다는것을 의미한다.

```
/home/juliet> cd cgi            프롬프트는 변경된 등록부를 보여 준다
/home/juliet/cgi> _
```

PWD는 그리 잘 리용되지 않는 변수로서 작업등록부가 변경될 때마다 재평가된다. C쉘의 prompt변수의 값주기에 변수 cwd를 사용하여 보면 재평가되지 않는다는것을 알수 있다.

PS1에서 사건번호의 사용(!)

bash도 리력기능을 지원하며 이것은 C쉘과 류사하다. 사용자는 사건번호가 할당된 지령에 대하여 재호출할수 있다. 기호 !는 이와 같은 사건번호를 나타내기 위하여 리용된다. 그러나 쉘이 그뒤의 문자렬들을 지령으로서 취급하지 않도록 \으로 의미해제시켜야 한다.

```
$ PS1 = '<\!> '            기호 \은 bash에서 필요하다
<508> _
```

여기에도 PWD변수를 포함시킬수 있다.

```
$ PS1='<\! $PWD> '
<509 /home/juliet/cgi> _
```

여기서는 외인용부호를 아무런 문제없이 사용하였다. 우의 실례에서 보는바와 같이 PS1에 대한 값주기지령은 리력목록에서 508번째 위치를 차지한다. 사건번호는 이전의 지령들을 호출하는데서 아주 유용하다.

PS1을 좀더 전용화하기

bash는 사용자의 프롬프트문자렬이 될수록 많은 정보를 포함할수 있게 하는 몇개의 확장문자렬들을 사용한다. 실례로 \h문자렬은 콤퓨터의 주콤퓨터이름을 보여 준다.

```
$ PS1="\h> "
satur> _            saturn은 콤퓨터의 이름이다
```

telnet를 리용하여 망에서 다른 콤퓨터에 접속하는 경우 사용자는 흔히 자기가 어디에 위치하고 있는지 혼돈할수 있다. 이 프롬프트는 항시적으로 자기가 어디에 있는가를 통지한다. 만일 프롬프트문자렬에 자기의 주콤퓨터이름을 설정하고 몇가지 다른 이름들을 보자면 원격콤퓨터에 가입되여야 한다.

아래에는 우리가 사용할수 있는 몇가지 확장문자렬(escape sequence)들을 보여 준다.

```
\s: 쉘의 이름
\t: 시:분:초형식의 현재시간(\T는 12시간형식)
```

\@: 오전/오후형식의 현재시간

\w: 홈등록부로부터 상대경로의 현재등록부

\u: 현재사용자이름

이 조종기호들을 여러개 결합하면 많은 정보를 포함하는 프롬프트문자렬을 얻을수 있다.

```
$ PS1="\h\@ \w>"
saturn 09:55am ~/project5/cgi> _
```

~(물결표)는 Bourne쉘을 제외한 모든 쉘들에서 홈등록부에 대한 생략된 형식이다. 여기서 사용자는 현재등록부(홈등록부아래의) project5/cgi에 위치하고 있다. ~는 17.8에서 구체적으로 론의한다.

그밖의 변수들

bash에서 사용되는 몇개의 다른 변수들이 있다.

HISTFILE, HISTSIZE, HISTFILESIZE는 17.5에서 론의된 지령리력기구에 의해서 리용된다.

BASH_ENV는 쉘이 호출될 때마다 실행되는 스크립트를 지적한다. 이 내용은 17.9에서 취급된다.

USER는 사용자의 가입이름을 보존한다. 이것대신에 LOGNAME변수를 리용할수도 있다.

17.4 별명

Bourne을 제외한 모든 쉘들에서는 별명으로서 자주 사용되는 지령들에 대하여 생략된 형식의 이름을 할당하는 **별명**(aliase)을 지원한다. 이것은 실례로 l이 ls -lids에 대한 별명이라면 앞으로는 ls -lids 지령대신에 l을 리용할수 있다는것을 의미한다. 사용자는 이 기능을 리용하여 정확한 선택항목을 가지고 호출되도록 현존지령을 재정의할수 있다. 실례로 많은 사람들은 ls -xF를 실행하도록 ls지령을 전용화한다. 이것은 alias문을 리용하여 진행할수 있다. C, Korn, Bash쉘들에서 정의상 약간 차이는 있으나 다음의 특징들은 모든 쉘들에서 일반적이다.

- 인수없이 사용되면 alias문은 이미 정의된 모든 별명들을 다 표시한다.
- alias문은 별명이름과 함께 사용되면 그의 정의로 된다.
- 별명은 그 이름과 함께 unaliased문을 리용하여 해제한다(별명정의가 해제된다).
- 별명은 재귀적특성을 가지고 있다. 이것은 a가 b라는 별명을 가지고 b가 c라는 별명을 가졌다면 a는 c로 된다는것을 의미한다.

그러면 3개의 쉘들에서 제공되는 별명화기능들에 대하여 취급하자. 이에 대한 설명은 Korn이나 bash보다 C쉘에서 더 구체적으로 진행된다. 그것은 Korn쉘과 bash쉘도 별명을 붙이기 위한 고급한 기능을 가진 쉘함수(19.10)들을 제공하기때문이다. C쉘은 함수들을 가지지 않으므로 별명은 쉘스크립트리용의 간접조작시간을 줄이기 위하여 지령렬을 생략하기 위한 유일한 방법이다.

17.4.1 C쉘에서 별명의 리용

C쉘은 별명에 대하여 지령행의 인수들을 포함하는 확장된 기능을 제공한다.

alias문은 두개의 인수(별명이름과 별명정의)를 리용한다. 아래의 명령문은 ls -1지령을 생략한다.

```
alias l ls -l              C쉘에서는 =기호가 없다
```

일단 이 방법으로 정의되기만 하면 ls -l지령을 실행시키기 위해서는 l을 호출해야 한다. 사용자는 그러한 이름으로서 외부지령 혹은 내부지령에 관계없이 리용할수 있다. 그러면 몇개의 파일이름을 가진 별명을 사용할 때 어떻게 되는가를 보자.

```
$ l relaydenied.html blankdel.sh
-rwx------    1 sumit    dialout       75 May  23 07:42 blankdel.sh
-rw-r--r--    1 sumit    dialout     7899 Jun  14 07:42 relaydenied.html
```

우의 지령은 얼핏 보기에 별명 l이 2개의 지령인수를 가지고 실행되였다고 생각할수 있지만 그렇게 생각해서는 안된다. 쉘은 별명을 그것이 나타내는 지령으로서 바꾸며 추가된 인수와 함께 전체 지령행을 실행시킨다. 그러나 C쉘의 alias는 인수들을 가지며 그것들은 별명안에서 특수한 위치파라메터들에로 읽혀 진다. 사용자는 적어도 아래의 두개 파라메터들을 알아야 한다.

* - 지령행의 모든 인수들을 표현한다.
\$ - 지령행의 가장 마지막인수를 표현한다.

이러한 표현식(expression)들은 리력기구로부터 나왔다. 리력기구에서는 이와 같은 표현식들을 리용하며 다만 이전 지령과 련관시킨다는 점을 제외하고는 서로 같은 의미로서 리용된다. 사용자는 파일탐색을 위한 별명을 만들기 위하여 마지막표현식을 사용할수 있다.

```
alias where 'find / -name \!$ -print'                !는 특수하다
```

이 별명을 리용하면 뿌리등록부로부터 시작하여 파일탐색을 진행할수 있다.

```
where pear1.jpg                pear1.jpg를 !$로 받는다
```

사용자는 별명을 만들 때 이전 지령의 가장 마지막인수로 그것을 교체하는것을 막기 위하여 파라메터 !$를 의미해제시켜야 한다. 이렇게 하면 인수 \!$는 현재지령의 마지막인수를 표현하게 된다. 만일 외인용부호가 모든 특수문자를 보호한다는 기성관념에 따른다면 착오가 생긴다. 여기서 이 외인용부호들은 문자 !를 보호하지 못하며 다만 \만이 보호한다. 사실상 인용부호는 전혀 필요 없다.

그러나 |와 같은 특수문자를 사용하여야 한다면 정확히 필요하다. 아래에는 한번에 한 페지씩 파일들을 목록화하여 보기 위한 하나의 별명을 보여 준다.

```
alias lsl 'ls -l \!* | more'
```

여기서는 별명이 여러개의 파일이름을 가지고 작업해야 하므로 \!*를 사용하여야 한다.

```
lsl 123.html thanks.ppt r*.html
```

만일 여기서 \!$를 사용하였다면 오직 마지막파일이름에 대한 목록을 보여줄것이다. 이 별명은 인수가 없이도 실행될수 있는데 이 경우에는 ls -l|more를 실행한다. 그러면 별명 where를 인수없이 실행시키면 어떻게 되겠는가? 아마도 탐색이 진행되지 않을것이라고 생각할것이다. 그러나 별명 where는 파일 where를 탐색한다. 그것은 쉘이 \!$을 지령행의 마지막단어(지령 혹은 별명 그자체)로서 해석하기때문이다.

C쉘은 앞으로 취급되겠지만 매개 인수들을 개별적으로 호출할수 있게 한다. 이 기능 역시 리력기구로부터 나왔으며 표현식 \!:n을 리용한다. 이 표현식에서 n은 하나의 수자 혹은 범위 지어는 몇개의 특수문자들까지도 표현할수 있다. 그러면 별명 where를 두번째 인수만을 받아 들이도록 수정하여 보자.

```
alias where 'find / -name \!:1 -print > \!:2'
```

여기서 보는바와 같이 \ !뒤로는 두점 :이 놓이며 또 그뒤의 수자는 지령행에서 인수의 위치를 표현한다. 사용자는 하나의 수값대신에 범위를 지적할수도 있으며 실례로 \ !:3-6은 3부터 6까지의 인수들을 표현한다. 이 수자화체계는 리력기구에 의하여 사용되며 이 문자들의 완전한 목록을 17.5에서 보여 준다.

별명에 대한 사용 역시 간단하다. 만일 긴 경로이름을 가진 등록부를 찾으려고 하는 경우에 아래와 같이 완성된 지령행 그자체에 별명을 붙일수 있다.

```
alias cddoc cd /usr/documentation/packages
```

또는 여러개의 렬로 실행파일들과 등록부들을 표시하기 위하여 ls지령 그자체를 재정의할수도 있다.

```
alias ls ls -xF
```

unalias지령을 리용하여 별명을 해제할수도 있다.

```
unalias where
```

alias지령에 인수가 없으면 정의된 모든 별명들의 목록을 보여 준다. 그러나 특정한 별명만 표시하기 위해서는 인수와 함께 그것을 리용할수 있다.

```
% alias where
find / -name !:1 -print > !:2
```

선택항목이 없이 C쉘을 리용하자면 별명을 조종하는데 익숙되여야 한다. 다른 사용자들은 위치파라메터 \!:1과 \!:2대신에 $1과 $2를 리용하는 쉘의 함수들을 배우기 위한 디딤돌로서 그것들을 간단히 리용할수 있다. 잘 믿어 지지 않겠지만 C쉘의 별명들은 \s를 충분히 사용해야 한다는 점을 제외하고는 if-then-else-endif형식의 조건문도 사용한다.

주해

외부지령이나 내부지령을 별명으로 변환하였을 때 본래의 지령을 실행시키자면 지령이름앞에 간단히 \기호를 선행시켜야 한다. 실례로 체계에 존재하는 어떤 지령을 탐색하기 위하여 \where로서 본래의 where지령을 실행시킬수 있다.

17.4.2 Korn쉘과 bash쉘에서 별명의 리용

Korn쉘과 bash쉘들도 alias지령을 사용한다. 사람들은 흔히 ls -l지령을 많이 리용한다. 이러한 지령에 대하여 사용자는 별명을 만들수 있다.

```
alias l='ls -l'                =기호가 필요하다
```

여기서 alias지령은 =기호를 요구한다. =기호의 량쪽에는 아무런 공백도 있어서는 안된다. 여기서 만일 대입하려는 값이 공백을 포함하고 있다면 인용부호로 둘러 막아야 한다. 이렇게 하면 ls -l지령을 간단히 l을 리용하여 실행시킬수 있다.

```
l                    이것은 ls -l을 실행시키는것으로 된다
```

사람들은 흔히 긴 경로이름을 가진 cd지령을 리용한다. 이러한 지령에 대하여 사용자들은 별명으로서 변환해야 한다는 느낌을 가질것이다. 별명으로서 변환한다면 다음과 같다.

```
alias intcd="cd /usr/spool/lp/interface"
```

그러면 이와 같은 별명들이 쉘스크립트에서와 같은 방법으로 지령행인수들을 받아 들이겠는가? 즉 몇개의 파일이름들과 함께 별명 1을 실행시킬수 있겠는가? 그러면 아래와 같은 지령을 실행시켜 보자.

```
$ l addbook.ldif apacheFAQ.html
-rw-r--r--    1 sumit    dialout     1555 Feb  8 14:03 addbook.ldif
-rw-r--r--    1 sumit    dialout   103485 Dec 13  1999 apacheFAQ.html
```

쉘은 먼저 별명 l을 ls -l로 확장하고 다음 2개의 인수들을 가진 ls -l을 실행시킨다. 비록 여기서 인수들을 가진 별명을 리용하였지만 사실은 그렇지 않다. Korn쉘과 bash쉘의 별명들에는 인수를 사용할수 없다. 그러나 어떤 지령이 자기의 마지막인수로서 파일이름을 사용한다면 현존지령들을 재정의하기 위하여 아래의 속성을 리용할수 있다. 이것이 우리가 다음으로 론하려고 하는것이다.

cp -i지령은 목적파일이 이미 존재하는 경우에 대화식으로 실행된다. 별명을 리용하면 cp지령을 항상 대화식으로 동작하게 만들수 있다. 또한 rm지령에 대해서도 이와 마찬가지이다.

```
alias cp="cp -i"
alias rm="rm -i"
```

우와 같이 정의하면 이러한 지령들이 호출될 때마다 기정적으로 항상 -i항목을 가진 지령이 실행되게 된다. 그러면 본래의 외부지령들은 어떻게 사용할수 있겠는가? 이 경우에는 반드시 지령앞에 \을 붙여야 한다. 실례로 파일 foo1을 foo2에 복사하기 위해서는 \cp foo1 foo2지령을 리용해야 한다. 여기서 사선기호 \ 는 별명을 무시하도록 한다.

사용자가 인수로서 이름을 가진 alias지령을 실행하면 그 이름에 대한 별명이 어떻에 정의되였는가를 표시한다.

```
$ alias cp
cp='cp -i'                bash의 출력은 약간 다르다
```

alias지령을 인수없이 실행시키면 모든 별명들이 다 표시되며 unalias문을 리용하여 별명을 해제할수 있다. 실례로 별명 cp에 대한 해제지령은 unalias cp 이다.

별명은 어떻게 만드는가에 따라서 큰 차이를 가진다. 여기에 저자가 사용하였던 몇개의 중요한 별명들을 보여 준다.

```
alias ..='cd ..'
alias ...='cd ../..'
alias dial='/usr/sbin/dip -v $HOME/internet/int3.dip'
alias dialk='/usr/sbin/dip -k'
alias mailg='fetchmail && elm'
alias mails='/usr/sbin/sendmail -q'
```

이것들중 대부분은 미리 정의된것들이며 그 나머지는 인터네트동작들을 고속화하기 위하여 저자가 정의한것들이다. 사용자는 지령 cd ..과 cd ../..을 자주 리용하는 경우 별명 ..과 ...이 아주 쓸모 있다는것을 알게 될것이다. 별명 dial은 모뎀으로 접속하기 위하여 intr.dip스크립트를 리용하며 ISP콤퓨터

에 접속한다. dialk는 반대로 그 접속을 해제한다. mailg는 전자우편봉사기로부터 우편을 얻으며 새로운 전자우편이 있다면 elm전자우편프로그람을 호출한다. mails는 봉사기에 대기우편을 보낸다. 이러한 모든 기능들은 이 책에서 론의된다. 다만 여기서는 기본적인 인터네트동작들이 몇개 안되는 별명들로서 어떻게 조종되는가 하는것을 보여 줄뿐이다.

주해

별명정의시에 주의할것은 별명을 너무 많이 붙이지 말아야 한다. 그렇게 되면 혼돈되기도 쉽고 또 기억하기도 힘들다. 더우기 별명들은 쉘함수들에 의하여 완전히 치환되며 이것들은 별명을 붙이기 위한 우월한 형식을 제공한다. 별명들은 처음에는 좋아도 결국에 가서는 쉘함수들을 사용하여야 할것이다.

17.5 지령리력

Bourne쉘의 중요한 약점은 지령을 다시 실행시키는 경우 그것을 다시 입력해야 한다는것이다. 다른 세개의 쉘들은 이전의 지령들을 불러 들여(이전 대화들에서 실행된것까지도) 필요하다면 그것을 편집하여 재실행시키게 하는 만능의 **리력**기능을 제공한다. 쉘은 매 지령에 대하여 **사건번호**(event number)를 할당하며 리력파일에 모든 지령들을(사용되는 쉘에 의존한다.) 보존한다.

history지령은 모든 지령의 사건번호를 보여 주는 리력목록을 표시한다. 지령은 !나 r와 같은 기호와 함께 이 사건번호를 리용하여 재호출된다. 이 파일의 최대크기와 이름은 두개의 특수변수들에 의하여 결정된다. C쉘과 bash쉘은 서로 류사한 기능들을 가지지만 Korn쉘은 일감에 대하여 여러가지 기호들을 사용한다. 리력기능들을 표 17-3에 요약하여 보여 준다.

17.5.1 C쉘과 bash에서 리력기능

alias와 같이 history지령도 하나의 내부지령이다. 기정적으로 이 지령은 자기목록안의 모든 사건들을 표시한다.

```
12 % history
     ....lines not shown ......
     8 grep "William Joy" uxadv??
     9 cd
    10 pwd
    11 doscp *.awk /dev/fd1135ds18
    12 history
```

매개 지령은 사건번호와 함께 표시된다. 사용자가 실행하는 모든 지령은 목록에 추가된다. 마지막지령은 history지령 그자체이다. 목록은 대단히 커질수 있기때문에 크기를 제한하기 위하여 수값인수를 리용하여 history지령을 호출할수 있다. 실례로 history 7지령은 마지막 7번째 지령만을 보여 준다.

표 17-3. 리력기능

csh, bash	ksh	의 미
history 12	history -12	마지막 12개의 지령을 표시한다
!!	r	이전 지령을 반복한다
!7	r 7	사건번호 7에 해당한 지령을 반복한다
!24:p	--	사건번호 24의 지령을 실행은 하지 않고 표시만 한다
!-2	r -2	마지막으로부터 두번째 지령을 반복한다
!ja	r ja	ja로 시작되는 마지막지령을 반복한다
!?size?	--	문자렬 size를 포함하고 있는 마지막지령을 반복한다
!find:s/pl/java	r find pl=java	pl을 java로 치환한후 마지막 find지령을 반복한다
^mtime^atime	r mtime=atime	mtime을 atime으로 치환한후 이전의 지령을 반복한다
!cp:gs/doc/html	--	doc를 html로 치환한후 마지막 cp지령을 반복한다
!! | sort	r | sort	이전 지령을 반복한 다음 sort로 관흐름처리를 진행한다
!find | sort	r find | sort	마지막 find지령을 반복한 다음 sort로 관흐름처리를 진행한다
cd !$	cd $_	등록부를 이전 지령의 마지막인수로 변경($_는 bash에 의해서도 사용된다.)한다
rm !*	--	이전 지령의 모든 인수들로부터 확장된 파일들을 삭제한다
vi !:2	--	이전 지령의 두번째 인수를 가지고 vi를 실행한다
grep http !30:4	--	사건번호 30의 네번째 인수를 가지고 grep http를 실행한다
more !:$	--	이전 지령의 마지막인수로서 more를 실행한다
!:0 foo	--	인수 foo를 가진 이전 지령을 실행한다
echo !$:h	--	이전 지령의 경로이름머리부를 표시한다
echo !$:t	--	이전 지령의 경로이름꼬리부(파일이름)를 표시한다
echo !$:r	--	이전 지령이 파일이름에서 확장자를 제거한다

이 쉘들은 지령들을 기억기와 파일의 량쪽에 다 보관한다. 또한 기억기의 리력목록의 크기를 결정하기 위한 변수와 보존된 파일의 최대크기를 결정하기 위한 변수를 사용한다. 이러한 변수들의 차이점을 아래에 보여 준다.

csh	bash	의 미
$HOME/.history	$HOME/.bash_history	리력목록을 포함하는 파일
savehist	HISTFILESIAE	보존될 리력목록의 크기
history	HISTSIZE	기억기에서 리력목록의 크기

C쉘과 bash는 리력기능을 관리하기 위한 동일한 지령들과 명령문들을 가지고 있다. 보통 사용자는 리력파일을 변경시키지 못한다. 그러나 목록의 기정크기를 변경시킬수 있다.

```
set savehist = 1000          리력에 보존한다-C쉘
HISTFILESIZE=1000           .bash_history에 보존한다-bash
```

마찬가지로 기억기에 보존되는 지령의 수를 설정할수 있다.

```
set history = 500                기억기에 보존한다-C쉘
HISTSIZE=500                    기억기에 보존한다-bash
```

그러면 마지막 15개의 지령들만을 표시하는 별명을 정의하여 보자.

```
alias h 'history 15'             C쉘
alias h='history 15'             bash
```

C쉘

리력기능은 history변수가 설정되지 않으면 자기의 동작을 원만히 수행하지 못한다. 이 경우에는 마지막지령만이 보존되며 그것도 기억기에서만 유효하다.

BASH쉘

C쉘에서와는 달리 여기서 리력파일은 HISTFUE변수에 대한 설정으로서 결정된다. 만일 설정되지 않으면 $HOME/.bash-history가 리용된다.

이전 지령들의 반복(!)

!지령은 이전의 지령들을 반복하기 위하여 사용된다. 이것은 리력기구의 주요기호이다. 이 기호는 정수 혹은 부수값, 문자렬 혹은 또 다른 !과 함께 리용될수 있다. 아래의 실례들에서 볼수 있는바와 같이 이 기호로서 시작하면 대단히 효률적이다.

마지막지령을 반복하자면 기호 !를 두번 리용해야 한다.

```
!!                     이전 지령을 반복한다
```

또한 사건번호와 함께 !를 리용하면 리력목록에서 임의의 지령을 반복실행시킬수 있다. !와 사건번호사이에는 그 어떤 공백도 있어서는 안된다.

```
% !11                     사건번호 11을 반복한다
11 doscp *.awk /dev/fd||35ds18
```

지령행이 표시되며 동시에 실행된다. 이와 같은 작업에서는 rm과 같이 틀린 지령이 실행될수도 있다. 이 경우 **변경자**(modifier: 연산자이기도 하다.) p(print)를 리용하여 지령의 실행이 없이 표시만 하게 할수 있다.

```
% !11:p
11 doscp *.awk /dev/fd1135ds18
```

doscp지령은 파일들을 디스크로부터 디스케트로 또는 그 반대로 복사한다. 여기서는 :p변경자를 리용하고 있다. 만일 이 지령을 실행시키고 싶다면 !!를 사용할수 있다.

사용자는 또한 관계주소를 리용하여 리력목록의 지령들을 재호출할수 있다. 즉 !:과 함께 부수값을 리용하여 마지막으로부터 두번째 지령을 쉽게 반복실행시킬수 있다.

```
!-2                     !와 -사이에 공백이 없다
```

사용자들은 자기가 최근에 리용한 2~3개의 지령을 제외한 나머지지령들에 대해서는 그의 사건번호를 기억하지 못할것이다. 그러나 최소로 지령이 어떤 특수한 문자나 문자렬로 시작되는가 하는것은 기억하고 있을것이다. 실례로 v로 시작하는 마지막지령이 vi지령이였다는것을 기억하고 있다면 !와 함께 v나 vi를 붙여 쓸수 있다.

```
!v                v로 시작하는 마지막지령을 반복한다
6
```

앞부분에서의 문자렬탐색이 자기의 목적에 맞지 않는다면 사용자는 매몰된 문자렬을 탐색할수도 있다. 이때에는 패턴의 량쪽에 반드시 ?를 붙여야 한다.

아래의 지령은 이전 지령에 대하여 임의의 위치에서 xvf를 탐색한다.

```
!?xvf?                 문자렬 xvf가 매몰된 마지막지령을 실행
```

tar지령은 플로피디스크에서 파일을 읽기 위하여 xvf항목을, 반대로 거기에 써넣을 때에는 cvf항목를 리용한다. !tar로써 마지막으로 사용되였던 tar지령을 재호출하는것은 자기원판에 파일을 덧쓰기할수도 있으므로 대단히 위험하다. 이렇게 놓고 보면 !?xvf는 확실히 안전한 방식이다. 이러한 항목들을 가진 틀린 지령이 실행될수도 있지만 실지로는 xvf항목을 리용하는 그밖의 다른 지령은 없다. 두번째 ?는 뒤에 [Enter]를 가진 문자렬에 대해서는 필요 없다. 매몰된 패턴에 대한 탐색은 Korn쉘에서는 불가능하다. 치환을 진행한다든가 또 경로이름의 요소들을 추출하기 위한 많은 지령들이 있다. 그것들은 다음에 취급된다.

주해

문자렬로 시작하는 마지막지령을 재호출하자면 !뒤에 그 문자렬을 놓는다. 지령행에 매몰된 문자렬을 가진 임의의 지령을 실행시키자면 문자렬의 량쪽을 ?기호로 둘러 막아야 한다. 두번째 형식은 여러개의 항목들로 지령들을 반복리용한 경우 대단히 유용하다. 즉 여기서는 지령인수에 대한 탐색이 필요된다. 만일 리력기능의 이러한 방법이 체계를 파괴시킬 경향이 있다고 생각되면 지령을 !!로서 반복실행시키지전에 단순한 표시만을 위해 :p를 리용하시오.

이전 지령에서의 치환(:s)

간단히 그리고 비대화식으로 편집을 진행할수 있게 하는 몇개의 변경자(modifier)들이 있다. 사용자는 변경자 :s를 리용하여 william을 Bill로서 치환한후에 이전의 grep지령을 재실행시킬수 있다.

```
!grep:s/william/Bill            :은 구분문자로서 동작한다
```

우의 치환은 전역적으로 진행되지 못하고 첫번째로 나타나는 문자렬에 대해서만 치환이 진행된다. 전역적인 치환을 진행하려면 g파라메터(sed에 의해서도 리용된다.)를 리용해야 한다. 다음의 지령은 doc의 모든 실체들을 bak로 전부 교체한후에 마지막으로 cp지령을 반복한다.

```
!cp:gs/doc/bak              Korn에서 불가능하다
```

만일 지름법을 리용한다면 이전 지령에서 문자렬에 대한 치환을 더빨리 진행할수 있다. 이 경우에는 !를 사용할 필요가 없으며 문자렬들에 대한 구분문자로서 반드시 ^(탈자기호)를 리용한다. 아래의 방법은 문자렬 bak를 doc로 치환하는것으로써 본래의 지령행을 부분적으로 회복한다.

```
^bak^doc                치환은 첫 실체에 대해서만 진행된다
```

이전 지령에 대한 인수들의 리용(!$, !*)

일반적으로 사용자들은 지령 mkdir foo로써 등록부를 만들고 cd foo지령으로써 그 등록부로 이행한다. C쉘과 bash쉘들은 지령행들을 생략할수 있는 특수한 패턴을 리용한다. 이러한 패턴으로서 !$이다. 이것에 대해서는 별명을 취급하면서 이미 론의하였다. 그러나 여기서는 이전 지령에서 사용된 등록부를 표현하는 생략기능으로서 !$를 리용한다.

```
mkdir programs
cd !$                   programs등록부에로 이동한다
```

또 다른 실례를 보자. 만일 쉘스크립트 cronfind.sh를 vi나 emacs로서 편집하였다면 간단히 다음과 같이 기입하여 이 파일을 실행시킬수 있다.

```
!$                cronfind.sh를 실행시킨다
```

이것은 vi나 emacs로 편집된 perl스크립트를 실행시키기 위한 좋은 방법이다. 사용자는 또한 패턴 !*을 리용하여 이전 지령의 모든 인수들을 표현할수 있다. 물론 별명에서도 이것을 리용하였다. 실례로 만일 지령 ls runj count.pl script.sh에 의하여 지적된 파일들이 있는가를 탐색하였다면 rm지령의 인수로서 !*(rm !* 은 실지로는 rm runj count.pl script.sh이다.)를 리용하여 그 파일들을 제거할수 있다. 이 지령은 Korn쉘에서는 쓸수 없다. 물론 여기에는 한계가 있다. 만일 지령 ls -l을 실행시키고 다음에 rm지령을 리용하였다면 그것은 인수로서 -l을 가지고 실행하였을것이며 물론 오유가 발생하였을것이다.

리력변경자의 사용(:n)

아래의 지령은 몇개의 파일들을 렬거한다.

```
ls runj count.pl script.sh add.sh binary.pl
```

사용자는 이전 지령의 개별적인 인수들을 꺼내기 위하여 단어변경자 :n을 리용할수 있다. n의 값은 0으로부터(지령) 시작하는 수값일수 있으며 범위 혹은 인수전체를 표현하기 위한 문자 *일수도 있다. 또한 첫 인수와 마지막인수를 표현하기 위하여 ^와 $을 리용할수 있다. 표 17-4는 vi에서 여러개의 파일들을 편집하자면 어떻게 해야 하는가를 보여 준다.

:n단어변경자는 마지막지령만이 아니라 사건번호에 해당한 모든 지령들에 다 적용된다. 사용자는 어떤 사건번호에 해당한 지령을 다른 사건번호에 해당한 지령의 인수들과 함께 실행시킬수 있다. 이전 지령들을 인수없이 호출하자면 변경자 :0을 리용해야 한다. 실례로 아래의 지령은 사건번호 47에 해당한 3개의 인수로써 com으로 시작하는 지령(compress)을 실행한다.

```
!com:0 !47:1-3
```

또한 다른 지령에서 인수들을 취하여 이전 지령을 실행시킬수 있다.

```
!:0 !-2:*               !:0은 인수를 가지지 않는 이전 지령이다
```

이것은 이전 지령행으로부터 지령만을 선택하며 마지막으로부터 두번째 지령의 인수들과 함께 그것을 실행시킨다. 이러한 특수기호들을 사용하여 얻는것은 시간이 걸린다. 그러나 이것은 C쉘을 오늘까지 사용하는 리유이기도 하다.

표 17-4. 리력변경자들의 리용(마지막지령: ls runj count.pl script.sh add.sh binary.pl)

리력변경자를 가진 지령	실 행
vi !:1	vi runj
vi !^	우와 같음(:이 필요 없다.)
vi !:2-3	vi count.pl script.sh
vi !:3-$	vi script.sh add.sh binary.pl
vi !:3*	우와 같음
vi !*	모든 인수들을 가진 vi지령
vi !:3-	vi script.sh add.sh(마지막인수를 제외한다.)
vi !$	vi binary.pl(:이 필요 없다.)

BASH쉘

!$외에도 bash는 지령의 마지막인수로 표시하기 위하여 $를 사용한다. 이 변수는 Korn 쉘에서도 사용된다.

경로이름변경자들(:h ,:t ,:r)

이 변경자들은 경로이름의 개별적인 부분들을 선택하는데 리용할수 있는 세개의 중요한 연산자들이다. 이전 지령에서 이러한 변경자를 사용하였다면 이 경로이름의 파생과 함께 같은(혹은 다른) 지령을 실행시킬수 있다.

먼저 :h(head)변경자를 보기로 하자. 이 변경자는 경로이름의 머리부를 추출한다. 즉 지령 ls /var/spool/lp/adrins으로 adrins등록부의 목록을 보았다면 이 변경자로 그 어미등록부에 대해서도 처리를 반복할수 있다. 또한 cd지령으로서 그 어미등록부로 이행할수도 있다.

```
ls !$:h              #ls/var/spool/lp지령을 실행한다
cd !$:h              #cd/var/spool/lp지령을 실행한다
```

:t(tail)연산자는 기본파일이름만을 선택한다. 이 변경자를 리용하여 사용자는 다른 등록부로부터 파일을 복사할 때 거기에 확장자를 붙일수 있다.

```
% cat /etc/skel/profile
...파일 내용을 현시...
% cp !$ !$ :t.txt                profile.txt로 복사한다
cp /etc/skel/profile profile.txt.               체계통보문
```

:r(root)연산자는 파일의 확장자를 떼버린 나머지부분만을 선택한다. 이 기능은 Java프로그람들을 실행시키는데 아주 쓸모가 있다.

```
% javac hello.java               hello.java를 콤파일한다
hello.java compiled successfully
% java !$:r              java hello를 실행시킨다
java hello
```

다른 또 하나의 변경자 :e도 있는데 이것은 파일이름에서 확장자만을 선택한다. C쉘과 bash에서는 이전 지령의 인수들과 함께 이 변경자들을 사용한다. 더우기 C쉘에서는 변수들과 함께 리용할수도 있다.

```
% set domain=planets.com
% echo $domain:e                C쉘에서만 유효하다
```

주해

Korn쉘에서는 지원하지 않는 C쉘과 bash의 4개의 중요한 리력기능들이 있다.

- 지령 !?xvf?와 같이 리력파일에서 매몰된 문자렬을 탐색할수 있다. Korn쉘에서는 지적된 문자렬로 시작되는 지령만을 탐색한다.
- 지령 !cp:gs/doc/back와 같이 이전 지령에서 전역적인 치환을 진행할수 있다. Korn쉘은 지적된 문자렬로 시작하는 이전 지령만을 반복실행하며 행의 첫 실체에 대해서만 치환을 진행한다.
- 이전 지령의 모든 인수를 의미하는 특수기호렬 !*과 선택적으로 인수들을 호출하는 변경자 :n을 리용할수 있다. Korn쉘에서는 마지막인수만을 호출할수 있다.
- Korn은 경로이름변경자들을 제공하지 못하지만 더 좋은 패턴탐색기구를 가지고 있다(19.8).

17.5.2 Korn쉘에서의 리력기능

인수없이 실행되는 history지령은 리력파일에 보존된 마지막 16개의 지령들을 보여 준다. 인수 -5로서 마지막 5개의 지령들을 볼수 있다.

```
$ history -5
36 exit
37 alias l='ls -l'
38 doscp *.pl a:
39 fc -l
40 history -5                목록을 표시하기 위하여 호출된 지령도 포함한다
```

지령들은 HISTFILE변수에서 정의된 파일에 보관된다. 만일 설정되지 않으면 $HOME/.sh_history가 사용된다. 사건목록에 보존된 지령들의 수는 HISTSLZE변수에 의하여 결정된다. 기정적으로는 128개의 지령들을 보존하며 사용자는 그것을 편리에 맞게 큰 수값으로서 설정할수 있다.

```
HISTSLZE=1200                1200개의 지령들을 기억할것이다
```

지령을 실행시킬 때마다 사건번호는 증가된다. 사건번호에 대한 지식은 이 번호를 참조하는것으로써 이전 지령들을 재실행시킬수 있기때문에 대단히 유용하다.

이전 지령들의 반복(r)

Korn쉘은 이전 지령을 반복하기 위하여 r지령을 사용한다. 인수없이 리용되면 r는 마지막지령을 반복한다.

```
r                이전 지령을 반복한다
```

인수로서 사건번호와 함께 r지령을 리용하여 다른 지령을 실행시킬수도 있다.

```
$ r 38                r와 38사이에 공백이 있다
38 doscp *.pl a:
```

반복적으로가 아니라 번갈아 두개의 지령들을 기동시킬 필요가 제기된다(실례로 C프로그람을 편집

하고 콤파일하기 위한 cc지령과 vi지령). 마지막지령이전의 지령을 실행시키기 위해 상대주소를 리용하여 번갈아 지령을 실행시킬수 있다.

```
r -2                교차기능
```

사건번호를 가진 지령호출은 일반적으로 이전 지령을 호출하는데서 특별히 리용되지 않는다. 사용자는 문자렬로서 지령을 호출할수 있으며 이때 지령은 그 문자렬로 시작되여야 한다. 실례로 v로서 시작하는 마지막지령이름이 vi지령이였다는것을 기억하고 있다면 이 지령을 호출하기 위하여 v 혹은 vi로서 r 지령을 사용할수 있다.

```
rv v                v로 시작되는 마지막지령을 반복한다
```

이전 지령에서의 치환(=)

사용자들은 때때로 이전 지령에 대한 치환이 진행된후에 다시 실행시켜야 할 필요성이 제기된다. 이 경우에 치환은 다음과 같은 형식으로 진행된다. 실례로 이전 cp지령을 반복하기전에 한개의 지령행에서 아래에서와 같이 문자렬 pl을 awk로 치환할수 있다.

```
r cp pl=awk
```

그러나 이 치환은 행에서 첫 실체에 대해서만 진행된다. 즉 둘 혹은 그이상의 치환은 진행되지 않는다(C쉘과 bash에서는 이 치환이 가능하다). 전체적인 치환 혹은 많은 편집이 요구되는 경우에는 직렬적인 지령편집기능을 사용해야 한다. 이것은 다음절에서 론의된다.

주해

만일 프로그람작성자이라면 아주 쓸모 있는 부분문자렬로 이전 지령들을 호출하는 기능을 알고 있을것이다(프로그람들을 반복적으로 편집하고 콤파일할 때). 실례로 지령 vi appl.java와 javac appl.java를 번갈아 리용하는 경우 처음에만 이 지령을 명백히 호출하고 그후부터는 Korn에서는 r v와 r j, 다른 쉘들에서는 !v와 !j를 번갈아 사용하여 실행시킬수 있다. 즉 그것들의 행번호를 기억할 필요가 없다.

이전 지령의 마지막인수의 사용($_)

흔히 동일한 파일에 대하여 여러개의 지령을 실행시키며 이 파일은 대체로 지령의 마지막인수이다. 사용자는 매번 파일이름을 지적하지 않고도 이전 지령의 마지막인수를 표현하기 위하여 패턴 $_을 리용할수 있다. 실례로 지령 vi applet.jara를 사용하였다면 그 프로그람을 콤파일하기 위하여 간단히 패턴 $_을 가진 javac지령을 호출할수 있다.

```
vi applet.java           $_는 applet.java를 표현한다
java $_             javac applet.java지령을 실행한다
```

vi foo지령으로서 편집된 쉘스크립트를 어떻게 실행시키겠는가? 간단하다. foo를 실행시키기 위하여 $_를 리용하시오.

```
$_                  마지막지령의 마지막인수를 실행시킨다
```

C쉘과는 달리 Korn쉘에서는 리력기능들이 대단히 많지는 못하지만 지금까지 론의된 지령들은 대부분의 일감을 처리하는데는 충분하다. 대신 행내편집기능을 제공하는것으로써 이 결함을 보상한다. vi나 emacs사용자들은 이전 지령을 반대로 호출하기 위하여 /patterm 혹은 [ctrl-r]pattern과 같은 지령들을 일반적으로 리용할수 있다. bash와 Korn은 이러한 기능을 지원하며 다음에 론의된다.

17.6 Korn쉘과 bash에서의 직렬지령편집

Korn과 bash는 vi 및 emacs지령행(현재지령과 그이전의 지령들)에 대한 편집기능을 제공한다. 이 기능을 **행내편집**(in-line editing)이라고 한다. 두 편집기들의 기본기능은 이 쉘들에 장비되여 있다. 이 기능을 리용하기전에 사용자는 아래의 지령들중 어느 하나를 실행시켜야 한다.

```
set -o vi            오직 하나만 선택하시오
set -o emacs
```

+o선택항목을 리용하여 방식절환을 진행할수 있다. 그러면 쉘의 vi편집기능들이 지령행편집에 어떻게 도움을 주는가를 보자. 만일 emacs를 알고 있다면 이 환경에 자기자신을 적응시키기 위한 방법을 알것이다. 표 17-5에서는 4장과 5장에서 설명한 편집지령들의 일부분을 보여 준다.

표 17-5. 행내편집지령들

vi지령	emacs지령	동 작
I	__	본문삽입
A	__	행의 끝에 본문추가
j	[Ctrl-n]	다음지령
k	[Ctrl-p]	이전 지령
l	[Ctrl-f]	오른쪽으로 한 문자
h	[Ctrl-b]	왼쪽으로 한 문자
w	[Alt-f]	한 단어앞으로
b	[Alt-b]	한 단어뒤로
0(령)	[Ctrl-a]	행의 시작
$	[Ctrl-e]	행의 끝
x	[Delete] 혹은[Ctrl-d]	문자지우기
dw	[Alt-d]	단어지우기
D	[Ctrl-k]	행의 끝까지 지우기
/pat	[Ctrl-r]pat	패턴 pat를 포함한 마지막지령
?pat	__	패턴 pat를 포함한 지령을 앞으로 탐색
n	[Ctrl-r]	초기탐색과 같은 방향으로 반복탐색
N	__	초기탐색과 반대방향으로 반복탐색
u	[Ctrl-_]	마지막편집명령의 취소

그러면 지령행에서 실수를 회복하여 보자. vi편집을 시작한 때부터는 입력한 이전의 본문에 대하여 거기로 유표를 이동하여 지울 필요는 없다. 먼저 [Esc]건을 눌러 vi의 지령방식으로 절환하시오. 이때로부터 사용자는 ^, $, b, e와 같은 조종지령들을 리용할수 있다. 필요하다면 반복인자를 사용할수도 있다.

사용자는 입력방식으로 들어 가기 위하여 리용되는 i, a, A 등과 같은 지령들을 호출할수 있으며 x지령으로써 문자를, dw지령으로써 단어를 지울수 있다. 사용자는 또한 p 혹은 P지령으로써 행의 임의의 위치에 단어를 놓을수 있다. 지령은 [Enter]건을 누르는것으로써 실행된다.

vi에서 유표를 우로 이동하였던 k지령이 여기서는([Esc]건을 누른후부터는) 이전 지령을 호출한다.

즉 반복인자가 적용된 지령 5k를 리용하여 현재지령을 제외한 가장 최근의 5개의 지령을 표시할수 있다. 만일 거기에 자기가 찾으려는 지령이 없는 경우 다음지령을 보기 위해서는 j지령을 리용하시오.

사용자는 이전 지령들을 호출하는 vi탐색기술을 리용할수 있다. 탐색이 반대방향에서 진행되여야 한다면(필요하다면 정규식과 함께) /pattern형식을 리용하여야 할것이다.

```
/find[Enter]              문자렬 find의 마지막출현을 탐색한다
/^find               마지막 find지령을 탐색한다
```

정방향탐색을 위해서는 ?pattern형식을 리용할수 있다. 또한 n을 누르는것으로써 탐색을 반복할수 있다. 지나치게 n을 계속 누르면 반대로 N지령이 실행된다.

행을 편집하고 그것을 다시 실행시키시오. 여기서 무엇을 알수 있는가? 행내편집기능은 C쉘을 아주 대중적인것으로 만든 리력변경자들의 결핍을 보상한다는것을 알수 있다. vi(혹은 emacs)에 숙련된 사용자들은 이전 지령들의 선택된 인수들과 함께 어떤 지령을 실행시키기 위하여 이 편집기능들을 간단히 리용할것이다.

만일 vi fool foo2 foo3 foo4지령이 실행되였다면 이 인수들중 일부 혹은 전부와 함께 compress지령을 쉽게 실행시킬수 있다. [Esc]건을 누르고 /vi로 지령을 탐색하시오. 다음 cw에 의하여 vi를 compress로 바꾸고 [Enter]건을 누르시오. 마지막인수를 지워야 한다면 4w로써 네번째 인수로 옮겨 가서 dw를 누르시오. 얼마나 간단한 작업인가?

Korn쉘

set -o외에도 Korn쉘은 다음의 두 변수들중의 어느 하나를 설정하여 지령행편집을 가능하게 할수도 있다.

```
EDITOR=/usr/bin/vi
VISUAL=/usr/bin/emacs
```

경로이름은 정확하지 않아도 된다. 즉 변수 VISNAL이 usr/bin/vi 혹은 abcd/vi로 설정되여도 일 없다. 쉘은 경로이름의 마지막에 놓인 문자렬 vi 혹은 emacs에 대하여 먼저 VISUAL변수를 보고 다음으로 EDITOR변수를 참조한다. 즉 이에 따라서 방식을 결정한다.

BASH쉘

bash에서는 우표이동건들을 리용하여 "DOSKEY"형식으로 이전 지령들을 재호출할수 있다. set -o지령은 이러한 방식에 대해서는 리용될수 없다. set -o vi를 리용하였다면 set +o vi로써 설정을 해제하시오.

17.7 파일이름완성하기

Korn과 bash는 **파일이름완성하기**(filename completion)라는 기능을 제공하며 최근 판본들에서는 이러한 기능이 더욱 확장되였다. 즉 다음과 같이 기능들을 지원한다.

- 지령의 인수로서 사용되는 파일이름의 완성하기
- 지령이름 그자체의 완성하기

이것은 사용자들이 완전한 지령이나 파일이름을 기입하지 않아도 된다는것을 의미한다. 그의 일부분만

을 기입하면 쉘이 그것을 확장하여 완성시킨다. 만일 emacs(5.10)에서 이와 류사한 기능을 리용해 본적이 있는 사용자라면 잘 알수 있을것이다. C쉘의 대부분의 판본들은 지령이 아니라 파일이름에 대해서만 완성하기기능을 지원한다. 세개의 쉘들에서 리용되는 완성하기작업에 필요한 건들을 표 17-6에서 보여 준다.

표 17-6. 파일이름과 지령이름의 완성하기문자들

	csh	ksh	bash
파일이름을 완성	[Esc]	[Esc]\	[Alt-/] 혹은 [Tab]
파일목록을 현시	[Ctrl-d]	[Esc]=	[Ctrl-x]혹은 [Tab][Tab]
지령이름을 완성	__	[Esc]\	[Alt-!] 혹은 [Tab]
지령목록을 현시	__	[Esc]=	[Tab][Tab]

17.7.1 Korn쉘에서 파일이름 및 지령완성하기

여기서는 우의 두 기능을 지원하는 ksh93판본을 참조하여 설명을 진행한다. 낡은 판본에 대해서는 지령완성기능을 쓸수 없다. 이 기능을 리용하기전에 사용자는 vi지령편집을 가능하게 하는 아래의 지령들중 어느 하나를 입력하여 완성하기기구를 능동으로 해야 한다.

```
set -o vi
EDITOR=vi
VISUAL=vi
```

완성하기기구를 리용하자면 두개의 지령이 반드시 필요하며 둘 다 [ESC]건을 요구한다. 실례로 p로 시작하는 아래와 같은 파일들을 가지고 있다고 생각하자.

```
$ ls -x p*
passwd              patlist              pattern.lst
planets.local.sam   planets.master.sam   planets.reverse.sam
problems.sam        profile.sam          stree.sam
```

만일 vi로 planets.master.sam파일을 편집하고 싶다면 vi를 입력한 다음 파일이름의 부분문자렬 pl 만을 입력하시오. 그리고 련속적으로 [ESC]\를 누르시오.

```
$ vi pl [ESC]\                planets로 된다
```

이 지령행은 즉시로 점을 포함하는 문자렬 vi planets.으로 확장한다. 쉘은 현재등록부에 pl로 시작되는 세개의 파일들이 있다는것을 발견한다. 이 세개의 파일이름들은 공통문자렬로서 planets.을 포함하고 있다. 그러면 이제 m을 기입하고 다시 [ESC]\를 리용하자.

```
$ vi planets.m[Esc]\                vi planets.master.asm으로 된다
```

세개의 파일들중 하나만이 공통문자렬 planets.뒤에 문자 m을 가진다. 여기로부터 사용자는 쉘이 지령행을 완성하였다는것을 알수 있다. 즉 3개의 문자들을 기입하고 Esc\를 두번 눌러서 파일이름이 완성되였다. 파일이름의 완성과정을 그림 7-1에서 보여 준다.

그림 17-1. Korn쉘에서 파일이름의 완성하기

파일완성하기기능을 리용한 또 다른 방법이 있다. 즉 지령행을 완성하기 위하여 쉘에 요구하는 방법이 아니라 파일목록에서 기입된 문자렬을 탐색하는 방법이다. 이 경우에 [Esc]=를 눌러야 한다. 이 기능을 리용하여 앞의 동작을 다시 해보면 다음과 같다.

```
$ vi pl[Esc]=
1)  planets.local.sam
2)  planets.master.sam
3)  planets.reverse.sam
113 $ vi pl
```

이때 쉘은 세개의 파일을 보여 주며 미완성된 지령행의 마지막문자에 유표를 놓는다. 다음 앞에서 한것처럼 그 과정을 반복하시오. 먼저 planet.으로 확장하기 위하여 [Esc]\를 누르고 m을 입력한 다음 다시 한번 [Esc]\을 누르시오. 여기서 일감은 완성된다.

ksh93에서 파일이름완성기능은 지령이름에 대해서도 적용되도록 확장되였다. uncompress지령으로써 압축파일을 해제한다고 하자. 첫 세개의 문자들을 입력하고 [Esc]\와 [Esc]=중 어느 하나를 리용하시오. 만일 unc[Esc]\을 리용하면 쉘은 unc를 uncompress로 확장할것이다. 지령은 5개의 건누름으로 완성된다. 만일 [Esc]=를 누른다면 그 패턴에 정합되는 단일한 파일이름을 보게 될것이다.

주해 Korn쉘의 판본을 알기 위하여 이 절의 서두에 반영된 기술을 리용하여 지령행편집을 가능하게 하고 다음으로 [ctrl-v]를 누르시오. 그러면 version 12/28/93와 같은 출력을 보게 될것이다. 이것은 ksh93이라는것을 말해 준다.

17.7.2 bash에서 파일이름 및 지령완성하기

Korn에서 사용된 기술은 bash에서도 동작한다. 그러나 bash는 주콤퓨터이름과 전자우편주소까지도 완성하여 주는 더 우월한 기능을 가지고 있다. 그러면 우리가 Korn에서 사용한것과 같은 파일을 리용해보자. 이 파일이름은 pl로 시작된다.

```
$ ls -x pl*
planets.local.sam      planets.master.sam      planets.reverse.sam
```

planets.master.sam파일을 편집하기 위하여 그의 유일한 부분문자렬인 pl을 입력하시오. 다음 [Alt-/]을 누르시오.

```
$ vi pl[Alt-/]                planets로 된다
```

bash는 pl을 planets.으로 확장하여 파일이름을 완성한다(점을 포함한다). pl로 시작하는 세개의 파일들이 모두 planets를 공통문자렬로 가지고 있으므로 이름을 좀더 완성하자면 문자 m을 입력하고 다음에 [Alt-/]을 눌러야 한다.

```
$ vi planets .m[Alt-/]                vi planets.master.asm으로 된다
```

planets문자렬뒤에 m을 가진 파일은 한개 파일뿐이며 목록에서 두번째 파일이다. 우리는 18개의 글자로 이루어 진 파일이름을 세개의 글자와 [Alt-/]을 두번 반복 리용하여 완성하였다. 파일이름의 완성과정을 그림 17-2에 보여 준다.

그림 17-2. bash에서 파일이름완성

사용자는 목록을 표시하는 ls지령을 리용하지 않고 그것을 bash자체가 표시하게 할수도 있다. 이때에는 다른 건렬 [Ctrl-x]/을 눌러야 한다. 앞에서 한것을 이 조종건을 리용하여 다시 해보자.

```
$ vi pl [Ctrl-x]/
planets.local.sam       planets.master.sam      planets.reverse.sam
```

```
$ vi pl
```

목록을 보면 다음번에 해야 할것을 상상할수 있을것이다. [Alt-/]을 눌러서 planet.으로 확장하고 다음에 m을 입력하고 다시 [Alt-/]을 누르시오.

bash는 지령이름완성하기도 지원한다. gunzip지령으로써 압축파일을 해제하자. 그의 첫 두 글자를 입력하고 [Alt-!]건을 누르시오.

```
gu[Alt-!]                [Shift]를 요구한다
```

쉘은 gu를 gunzip로 확장한다. 만일 변수 PATH에 반영된 등록부들에서 입력된 문자렬과 정합되는 모든 지령들을 보고 싶다면 문자렬을 입력한 다음 [Tab]건을 두번 누르시오.

참고

bash에서는 파일이름 혹은 지령완성하기를 위하여 [Tab]건을 리용할수 있다. 문자렬의 일부분을 입력하고 [Tab]건을 한번 누르면 파일이름이나 지령은 가능한껏 완성된다. 목록을 표시하기 위해서는 [Tab]건을 두번 누르시오. 이것은 set -o vi에 의한 행내편집이 가능한가 불가능한가 하는것에 관계없기때문에 제일 좋은 방법이다.

C쉘

C쉘의 대부분의 판본들은 파일이름완성하기는 지원하지만 지령이름완성하기는 지원하지 않는다. 지령은 그것이 파일로 취급되도록 현재등록부에 존재해야 한다. 만일 리용하려는 판본이 파일이름완성기능을 지원한다면 set filec로 그것을 가능하게 해야 한다. 이와 같은 완성에 대해서는 [Esc]건을 리용하며 목록을 표시하기 위해서는 [Ctrl-d]를 리용한다.

17.8 기타 기능

쉘의 초기화스크립트들을 론의하기전에 때때로 리용할 필요가 있는 몇가지 기능들을 보기로 하자.

- 파일들이 우연히 재쓰기되는것을 방지한다. 이 기능을 **덧쓰기무시**(noclobber)라고 부른다. 일단 설정하면 쉘의 기호 >와 >>에 의하여 덧쓰기되는것으로부터 자기 파일들을 보호할수 있다.
- 또한 [Ctrl-d]을 눌렀을 때 체계에서 탈퇴(logout)되지 않도록 한다. 사용자들은 흔히 표준입력을 끝낼 작정으로 [Ctrl-d]를 무심결에 누르지만 그렇게 하면 체계에서 탈퇴되게 된다. **파일끝무시**(ignoreeof)기능은 이것을 막기 위한 안전자물쇠를 제공한다.
- 홈등록부의 생략표현으로서 물결표(~)를 리용한다.

이 기능들은 Bourne쉘에서는 불가능하며 또한 다른 쉘들에서는 조금씩 다르게 실현된다.

17.8.1 C쉘에서 덧쓰기무시와 파일끝무시

C쉘은 파일에 대한 우연한 덧쓰기를 방지하기 위하여 noclobber인수를 가진 set문을 리용한다.

```
set nodobber                파일들은 >에 의하여 더는 덧쓰기되지 않는다
```

만일 이미 존재하는 파일 foo로 지령의 출력자료를 내보낸다면 통보문이 제시된다.

```
foo: File exists.
```

이 보호기능을 무효로 하기 위해서는 기호 >다음에 !를 리용해야 한다.

```
head -5 emp.lst >! foo
```

ignoreeof인수를 리용하여 체계에서의 우연적인 탈퇴를 막을수 있다.

```
set ignoreof               [Ctrl-d]를 눌러도 탈퇴되지 않는다
```

이렇게 되면 대화를 끝내기 위하여 [ctrl-d]를 누르는 경우 쉘은 다음과 같은 통보문을 내보낸다.

```
use "logout" to logout.
```

사용자는 대화에서 탈퇴하기 위해서는 C쉘의 logout지령을 리용해야 한다. 또는 exit지령을 쓸수도 있다.

17.8.2 Korn과 bash에서 덧쓰기와 파일끝무시

set문은 -o선택항목을 지원한다. 파일내용의 우연한 덧쓰기를 막기 위해서는 아래의 방법과 같이 인수 noclobber을 리용해야 한다.

```
set -o noclobber               파일들이 기호 >로서 더는 덧쓰기되지 않는다
```

이렇게 하면 사용자가 이미 존재하는 파일 foo로 출력절환을 진행하는 경우 쉘은 다음과 같은 통보를 내보낸다.

```
bash: foo: cannot overwrite existing file          bash
ksh: foo: file already exists                      Korn
```

이 보호기능을 무효로 하기 위해서는 기호 >다음에 |를 리용하시오.

```
head -5 emp.lst >| foo
```

[Ctrl-d]에 의한 우연한 탈퇴는 set -o와 함께 ignoreof인수를 리용하는것으로써 보호된다.

```
set -o ignoreof               [Ctrl-d]로는 탈퇴되지 않을것이다
```

사용자가 대화를 완료하기 위하여 [Ctrl-d]를 리용하면 쉘은 다음의 통보문을 내보낸다.

```
use 'exit'to terminife this sehll
```

이 경우에는 exit지령을 리용하여 대화를 완료해야 한다. 즉 [Ctrl-d]로 반복실행하지 않는한 탈퇴하지 못한다. set선택항목은 set +o에 의하여 해제된다. 덧쓰기무시기능을 회복하자면 set +o noclobber를 리용하시오. set의 특수한 선택항목들의 완전한 목록은 지령 set -o 혹은 set +o를 인수없이 실행시켜 볼수 있다.

BASH쉘

bash는 가입대화를 완료하기 위한 C쉘형식의 logout지령을 지원한다.

17.8.3 물결표에 의한 치환

~는 홈등록부의 생략형식으로서 Borne을 제외한 모든 쉘들에서 리용된다. 기호 ~가 아래와 같이 가

입이름앞에 놓이면

```
cd ~juliet                이것은 cd $HOME/juliet와 같다
```

은 juliet의 홈등록부로 바뀐다. 만일 $HOME의 값이 /home/juliet/html이라면 이것은 지령 cd ~juliet에 의해 이행하는 등록부이다. 흥미 있는것은 기호 ~자체가 홈등록부라는것이다. 만일 juliet로서 가입하였다면 cd ~/html을 리용하여 홈등록부아래의 html등록부를 호출할수 있다.

우리가 흔히 찾는 .profile과 같은 구성파일은 $HOME/.profile 혹은 ~/oprofile로서 참조할수 있다.

Korn쉘

BASH쉘

지령 cd -를 리용하여 사용자는 현재등록부와 그이전에 마지막으로 보았던 등록부사이로 쉽게 절환할수 있다. 이것은 많은 텔레비죤원격조종기들에서의 현재통로와 마지막으로 본 통로들사이를 절환시키는 단추와 같은 기능을 수행한다. 이 지령의 사용법은 다음과 같다.

```
[/home/image] cd /bin      등록부 /home/image로부터 /bin으로 이행한다
[/bin] cd -                등록부 /home/image로 이행한다
/home/image                쉘은 이것을 표시한다
[/home/image]                현재등록부의 PS1
```

17.9 초기화스크립트

우리는 많은 환경변수들에 값을 대입하였으며 또 별명을 정의하고 set선택항목을 리용하였다. 이러한 설정들은 그 대화에 대해서만 적용할수 있으며 사용자가 체계에서 탈퇴하면 기정값으로 회복된다. 이것을 영구화하자면 체계시동스크립트들에서 그러한 설정을 진행해야 한다.

모든 쉘들은 Windows의 AUTOEXEC.BAT파일과 같은 시동스크립트를 적어도 한개 리용한다. 이 스크립트들은 사용자의 홈등록부에 있으며 사용자가 체계에 가입할 때 실행된다. 일부 쉘들은 체계에서 탈퇴하기전에 일부 개별적인 파일을 실행시킨다. ls -a로서 자기의 홈등록부를 보시오. 여기서 아래와 같은 파일을 보게 될것이다.

- .profile(Bourne쉘)
- .login, .cshrc, .logout(C쉘)
- .prifile, .kshrc(Korn쉘)
- .bash_profile(혹은 .profile.bash_login), .bashrc, .bash logout (bash)

여기서 하나의 스크립트는 다음의 세 부류가운데서 어느 한 부류에 속한다(표 17-1의 마지막 세개의 항목을 보시오).

- 가입스크립트: 이것은 사용자가 체계에 가입할 때 실행되는 시동스크립트이다. 가입스크립트로서 C쉘은 .login을, Bourne과 Korn쉘은 .profile을, bash는 .bash_profile을 리용한다.
- 환경스크립트: 이것은 가입쉘로부터 보조쉘이 기동될 때 실행된다. 일반적으로 rc스크립트라고 말하며 Bourne쉘을 제외한 모든 쉘들이 각각 한개씩(.cshrc, .kshrc., .bashrc) 가진다. 정확히 말하면 Korn쉘의 rc스크립트가 반드시 .kshrc로 되는것은 아니지만 흔히 이 이름을 리용한다.
- 탈퇴스크립트: C쉘과 bash만이 사용자가 체계에서 탈퇴하기전에 탈퇴스크립트(.logout

와 .bash_logout）를 실행시킨다.

다음절들에서 이러한 파일들에 대하여 론의한다. 사용자는 그것들안에 놓을수 있는 쓸모 있는 선택항목들을 결정하게 될것이다. 이에 앞서 사용자는 체계가입시에 우리가 가지게 되는 쉘과 스크립트들을 실행시키는 쉘사이의 차이점을 알아야 한다. 또한 보조쉘들을 만들지 않고 이 스크립트들을 실행시키는 방법에 대해 배워야 한다.

17.9.1 점(.)과 source지령

쉘스크립트를 실행시킬 때 현재쉘은 스크립트내의 지령들을 실행시키는 보조쉘을 자기자체내에서 생성한다. 스크립트내에서 진행된 변수대입과 등록부변경은 스크립트를 기동시킨 쉘에서는 보이지 않는다. 그것은 새끼쉘에서 만들어 진 변화들을 어미쉘에서는 볼수 없기때문이다(10.2).

그러나 가입쉘이 초기화파일들을 실행시킬 때에는 이와는 전혀 다르다. 이 파일들에서 정의되는 변수들은 가입쉘에서 보인다. 즉 등록부의 변경은 스크립트실행이 끝난후에도 유지된다. 여기로부터 우리는 이러한 스크립트들이 보조쉘이 없이 가입쉘에 의하여 실행된다는것을 알수 있다.

보조쉘을 만들지 않고 임의의 쉘스크립트를 실행시킬수 있는 두개의 지령이 있다(.과 source지령들). C쉘은 source를, Bourne과 Korn은 .을, bash는 둘 다 리용한다. 사람들이 생각하는것과는 반대로 시동스크립트를 변경시킨 경우에 체계에서 탈퇴하였다가 다시 가입할 필요가 없다. 즉 아래와 같은 방법으로서 스크립트를 실행시키시오.

```
. .profile             Bourne과 Korn쉘
source .login          C쉘
. .bash_profile        bash
source .bash_prfile    (bash-과 source 둘 다 리용)
```

일단 이와 같은 방법으로 실행시키면 그 변화들은 현재쉘에서 유효하게 된다. 사용자들은 뒤장에서 쉘함수들을 포함하는 파일을 실행시키기 위하여 이 기능을 리용할것이다.

주해

.과 source지령은 보조쉘의 사용이 없이 스크립트를 실행시킨다. 그것들은 또한 스크립트가 실행허가를 가질것을 요구하지 않는다. 대부분의 이러한 지령들은 현재등록부가 PATH변수에 반영되여 있지 않다면 례외적인 동작패턴을 나타낸다. 즉 파일에 대하여 상대 혹은 절대경로이름이 리용되지 않으면 현재등록부에서의 파일탐색이 실패한다. 만일 .이 PATH변수에 반영되지 않으면 지령 . .pr ofile은 실행되지 않을것이며 반드시 . ./.profile을 사용하여야 한다. 그렇지만 C쉘은 정상적으로 동작한다.

17.9.2 대화형쉘과 비대화형쉘

모든 쉘들은 두가지 형태 즉 대화형과 비대화형으로 분류할수 있다. 체계에 가입하면 사용자는 프롬프트를 표시하고 사용자의 질문을 대기하는 **대화형쉘**을 보게 된다. 대화형쉘에서는 일감조종, 지령행편집 및 리력기능이 가능하다. 또한 비대화형쉘들과는 다른 방식으로 신호들에 응답한다. vi와 emacs와 같은 일부 UNIX지령들은 대화형보조쉘의 프롬프트에서 작업하게 하는 쉘탈퇴기능들을 가지고 있다.

사용자가 이러한 쉘로부터 쉘스크립트를 실행시키면 **비대화형쉘**이 호출된다. 대화형쉘의 많은 파라메터들이 비대화형쉘로 넘겨 지지 않기때문에 그것들을 유효하게 하려면 개별적인 배렬을 만들어야 한다. 대화형쉘은 시동(가입)스크립트와 개별적인 rc(환경)스크립트를 실행한다. 비대화형쉘이 대화형쉘로부터

실행될 때 그것은 rc스크립트만을 실행한다. 이 rc스크립트는 보조쉘의 모든 설정을 포함한다.

그러면 Bourne쉘에서 리용되는 .profile을 론의하자. 자기가 리용하는 쉘이 Bourne쉘이 아니라고 해도 이 부분에서는 다른 쉘들에 대한 개념적인 내용들을 론의하기때문에 반드시 읽어야 한다.

주해

환경변수들과 별명(set선택항목들 물론)사이에는 일정한 차이는 있어야 한다. 환경변수들은 모든 보조쉘들에서 유효하다. 그러나 별명들과 set선택항목들은 보조쉘들에서는 자동적으로 유효하게 되지 않는다. 그것들은 항상 rc파일들에 놓여 지며 대화형쉘들에 전달된다. 앞으로 보게 되겠지만 별명은 C쉘스크립트들에서 유효하다.

17.9.3 Bourne쉘에서의 초기화스크립트(.profile)

ls -a지령에 실행시켜 자기의 홈등록부에 .profile이 있는가를 보시오. 이 가입스크립트는 사용자가 만들 때 홈등록부에 추가된다. 만일 거기에 파일이 없는 경우 사용자는 이러한 스크립트를 만들수도 있다. 사용자가 체계에 가입할 때 쉘은 아래의 두 파일을 실행시킨다.

- 전역적인 초기화파일 /etc/profile: 이 파일은 모든 사용자들에 대하여 공통적으로 쓰이는 변수들을 설정하며 지령들을 내보낸다.
- .profile: 이 파일은 사용자에 의하여 만들어 진 설정들을 포함한다.

파일 .profile은 사용자의 요구에 따라 매우 커질수 있다. 그러나 요약하면 초기의내용을 수행해야 한다.

```
$ cat .profile
# User $HOME/.profile ? commands executed at login time
MAIL=/var/mail/$HOME                         # mailbox location
IFS=
PATH=$PATH:$HOME/bin:/usr/ucb:.
CDPATH=.:..:$HOME
PS1='$'
PS2=>
TERM=ansi
MOZILLA_HOME=/opt/netscape;export MOZILLA_HOME
calendar
mesg y
stty stop ^S intr ^C erase ^?
```

일부 체계변수들은 이 스크립트에서 할당된다. 뒤의 명령문들은 그 파일이 기호에 맞게 편집되였다는것을 나타낸다. PATH변수는 세개이상의 등록부들을 포함하고 있다

CDPATH변수는 홈등록부와 어미등록부아래의 등록부들을 호출할수 있게 한다. 또한 calendar지령은 자기가 계약했던것을 상기시키기 위한 좋은 수단이다. mesg y는 talk지령(11.2)을 리용하는 사람들로부터 통보문들을 받아 들이기 위한 자기의 의지를 반영한다. 또한 이 스크립트에서 일부 stty설정을 진행하였다.

스크립트에 정의된 대부분의 변수들은 /etc/profile(쉘이 .profile을 실행시키기전에 실행하는)로부터 계승된다. 변수 MOZILLA_HOME은 이 스크립트에서 정의되며 Netscape를 정확히 실행시키자면 이 변수를 반출(export)시켜야 한다(PATH와 같은). 다른 변수들은 이미 반출되였다. 그러므로 그의 값을 변경시켰다고 하여 다시 그것들을 반출시켜서는 안된다.

Bourne쉘은 이 장에서 론의되는 별명이나 개선된 특징들중 일부를 지원하지 않으므로 rc스크립트가 필요 없다. .profile에서 정의된 모든 변수들은 보조쉘들에서 쓸수 있도록 스크립트 그자체에서 반출되여야 한다. Bourne은 쉘프로그람작성에 대한 지식을 가지고 그 동작이 평가된다고 하여도 탈퇴스크립트를 실행시키지 않는다.

주해

.profile은 /etc/profile이 처리된 다음에 실행된다. /etc/profile은 모든 사용자들에 대한 종합적인 가입스크립트이다. 즉 공통적인 환경설정들은 모든 사용자들에 의하여 리용되도록 관리자에 의하여 /etc/profile에 보존된다. 만일 체계가 /etc/skel등록부를 가지고 있다면 그 등록부안에 파일 .profile(혹은 local.profile)이 있을것이다. 이 파일은 useradd지령에 의하여 사용자의 홈등록부에 추가되는 파일이다.

17.9.4 C쉘에서의 초기화스크립트(.cshrc, .login, .logout)

C쉘은 시동명령들에 대하여 .profile을 리용하지 않는다. 대신에 두개의 다른 파일들을 리용한다. 파일 ~/.logout는(존재한다면) 체계에서 탈퇴하기전에 실행된다. 사용자가 가입할 때 쉘은 아래의 순서로 세개의 스크립트를 실행시킨다.

- 전역초기화파일: 이 파일은 /etc/login 혹은 /etc/.login(Solaris) 일수 있다. 모든 사용자들에 의하여 실행되는 공통적인 명령들은 다 여기에 놓인다. 이 파일은 모든 체계들에서 존재하지 않을수도 있다.
- ~/.cshrc 파일: C쉘이 기동될 때마다 실행되는 명령들을 포함한다.
- ~/.login파일: 사용자가 가입할 때에만 실행된다.

여기서 홈등록부의 생략기호 ~를 리용하였다. 또한 그것은 Korn과 bash에서의 동작패턴과 다르다. 먼저 쉘은 등록부/etc에서 가입파일을 실행시킨다. 다음 그것이 가입쉘인가 하는것을 결정한후에 ~/.login을 기동시킨다. ~/.cshrc이 호출되는 모든 보조쉘들에서 실행된다면 반대로 스크립트 /etc/login과 ~/.login은 오직 한번만 실행된다.

C쉘의 환경파일들은 간단한 동작형태를 따른다. 사용자가 대화형쉘을 리용하는가 혹은 비대화형쉘을 리용하는가에 관계없이 파일 .cshrc는 항상 실행된다(Korn쉘은 다르게 동작한다). .login은 TERM과 같은 환경변수설정과 한번만 실행될 필요가 있는 명령들을 포함하고 있다.

```
calendar
mesg n
stty stop ^S intr ^C erase ^?
setenv MOZILLA_HOME /opt/netscape              Netscape에 의해 요구됨
setenv TERM vt220
```

변수 MOZILLA_HOME과 TERM은 setenv명령에 의하여 환경변수로서 지적되여야 한다. 보조쉘들에 자동적으로 계승되지 않은 특수한 C쉘변수들은 파일 ~/.cshrc에 정의되여야 한다.

```
set prompt= '[\!]'
set path=($path /usr/local/bin)
set history=500
set savehist=50
set noclobber
set ignoreeof
```

변수 prompt, path, history들은 C쉘(국부)변수들이며 그 값들은 새끼쉘들에서 유효하게 된다. 더우기 noclobber와 ignoreeof설정은 보조쉘들에서 반출되지 않으며 새끼쉘이 만들어 질 때마다 사용되여야 한다. 별명정의가 새끼쉘에까지 계승되지 않기때문에 그것들은 파일~/.cshrc에 놓여 져야 한다.

```
alias l ls -l
alias ls-l ls -l
alias h "history | more"
alisas ls ls -aFx
alias h history
alisa rm rm -i
```

여기로부터 우리들은 스크립트 ~/.cshrc이 ~/.login보다 더 빨리 구성된다는것을 알수 있다. 대부분의 설정들은 대화형쉘들에서만 리용되거나 관계된다. 그러나 그것들은 여전히 쉘스크립트들에서 유효할것이다.

17.9.5 Korn쉘에서의 초기화스크립트(.profile, ENV)

Korn쉘이 Bourne의 상위모임이므로 앞의 17.9.3에서 론의된 모든것들은 Korn쉘에서도 적용된다. 그러나 Korn은 기능적으로 풍부하며 자기자체의 환경스크립트(rc)를 요구한다. 이 스크립트이름은 임의로 할수 있지만 ENV변수에 의하여 설정된다. 대부분의 사용자들은 C쉘로부터 Korn쉘로 이동하므로 이 스크립트는 흔히 파일 ~/.kshrc이다. 사용자가 체계에 가입할 때 3개의 스크립트가 차례로 실행된다.

- 전역초기화파일 /etc/profile: 모든 사용자들에게 공통적인 설정들을 포함한다.
- 사용자자신의 ~/.profile
- ENV변수에서 정의된 파일: 이것은 보통 ~/.kshrc이다.

Korn의 동작형태는 Bourne에서와 약간 다르다. 그렇다고 하여 놀랄것은 하나도 없다. 즉 .profile에 한개의 지령이 더 추가되였을뿐이다.

```
export ENV=$HOME/.kshrc
calendar
mesg n
stty stop ^S int ^C erase ^?
```

여기서 우리들은 특별한 변수 ENV를 볼수 있다. 이 변수는 Korn에 의하여 리용되는 파일 .profile에 반드시 반영되여야 한다. ENV변수는 두가지 목적에 리용된다.

- 가입시에 그리고 보조쉘을 호출할 때 실행되는 파일을 정의한다.
- 환경파일을 실행시킨다.

파일 .profile이 가입시에만 실행되므로 한번만 실행될 필요가 있는 명령들을 이 파일에 놓는다. 나

머지명령들은 환경파일에 놓아야 한다. 아래에 파일 .kshrc의 입력항목들을 보여 준다.

```
MAIL=/var/mail/$LOGNAME                #mailbox location
PATH=$PATH:$HOME/bin:/usr/dt/bin:.
CDPATH=.:..:$HOME
PS1= '[$PWD]'
PS2=>
HISTSIZE=2000

set -o noclobber
set -o vi

alias h= "history -10"
alias cp= "cp -i"
alias rm= "rm -i"
```

엄밀히 말하여 .kshrc에서 환경변수설정은 .profile에서도 진행할수 있다. 이 경우에는 프로필들이 새끼쉘들에 의하여 실행되지 않기때문에 대부분의 변수들은 명백히 반출되여야 한다. 그러나 set -o나 alias지령들은 반드시 파일 .kshrc에 있어야 한다. C쉘에서와는 달리 Korn쉘별명들은 대화형쉘들에서만 유효하며 쉘스크립트에서는 그렇지 못하다(지령 alias의 -x선택항목 역시 이것을 허락한다. 그러나 이 특징은 ksh93에서 없어 졌다).

참고

더 좋기는 개별적인 파일에 례하면 ~/.aliases에 별명들을 놓아야 할것이다. 쉘스크립트들에서 이와 같은 별명들을 리용해야 한다면 스크립트의 첫 시작부분에 명령문 . ~/.aliases을 놓으시오. 이것은 사용자가 Korn쉘의 어느 판본을 리용하는가에는 무관계하다.

17.9.6 bash에서의 초기화스크립트(.bash_profile, .bash_logout, .bashrc)

bash 역시 Bourne의 상위모임이며 따라서 앞의 17.9.3에서 론의된것들은 이 쉘에서도 적용된다. 또한 bash는 C쉘에서와 같은 초기화파일들의 정교한 묶음를 지원한다. 사용자가 체계에 가입할 때 아래와 같은 순서로 스크립트들이 실행된다.

- 전역적인 초기화파일 /etc/profile: 이 파일은 모든 사용자들에게 공통적인 설정들을 포함하고 있다.
- 사용자자체의 가입스크립트: bash는 3개의 파일 즉 ~/.bash_profile, ~/.bash_ login, ~/.profile들을 탐색한다. bash는 이 파일들중 어느 하나라도 발견되면 그외의 다른것들을 무시한다.
- 변수 BASH_ENV에 설정된 환경파일: 이것은 프로필에 명백히 지적되여 있는것을 전제로 한다. 이 파일은 일반적으로 ~/.bashrc이다.

bash는 또한 사용자가 체계를 탈퇴하기전에 파일 ~/.bash_logout를 실행한다. 이것을 론의하기 위해서 bash의 가입스크립트를 .bash_profile이라고 가정하자. C쉘과 Korn에서와 같이 아래의 명령문들을 가입시에만 실행되는 파일 .bash_pr ofile에 놓으시오.

```
stty stop ^S int ^C erase ^?
mesg n
```

```
export BASH_ENV=$HOME/.bashrc
. ~/.bashrc
```

여기서 변수 BASH_ENU는 환경파일을 정의한다. 이 항목이 없으면 보조쉘들도 파일 .bash_profile을 실행시키므로 거기에 놓아야 한다. 즉 가입쉘은 .bash_profile에 특별히 지적되지 않는한 그것을 자동적으로 하지는 않는다. 환경파일은 set -o명령들 그리고 별명과 같은 모든 환경설정을 포함하여야 한다.

```
MAIL=/var/spool/mail/$LOGNAME                #mailbox location
PATH=$PATH:$HOME/bin:/usr/X11R6/bin:/usr/lib/java/bin:/opt/kde/bin:.
CDPATH=.:..:$HOME
PS1="\u@\h:\w>"
PS2='>'
HISTSIZE=1000
HISTFILESIZE=1000
TERM=linux
set -o emacs
set -o ignoreeof

alias dial='/usr/sbin/dip -v $HOME/internet/int3.dip'
alias dialk='/usr/sbin/dip -k'
alias mailg='fetchmail && elm'
alias mails='/usr/sbin/sendmail -q'
```

MAIL변수의 값을 보면 이것이 Linux체계라는것을 알수 있다. 여기서 변수 PS1은 사용자이름, 주콤퓨터이름, 현재등록부를 반영한다. 별명정의는 이미 17.4.2에서 설명되였다.

엄밀하게 말하면 환경변수들은 .profile(혹은.bash_profile)에 할당할수도 있다. 그러나 가입스크립트가 보조쉘들에 의하여 호출되지 않기때문에 이미 반출되지 않은 변수들에 대해서는 명백히 반출시켜야 한다. 변수 TERM과 PATH는 여기서 재할당되며 반출시킬 필요가 없다. 그러나 HISTSIZE와 HISTFILESIZE는 아마도 반출되여야 할것이다.

별명들과 set선택항목들은 반출되지 않으므로 .bashrc에 정의되여야 한다. C쉘과는 달리 bash별명들은 대화형쉘들에서만 유효하며 쉘스크립트들에서는 그렇지 않다. 또한 17.9.5의 《참고》에서 서술된것처럼 개별적인 파일 .aliases에 별명들을 정의하는 기술조차 리용할수 없다.

참고

가입탈퇴전에 실행하고 싶은 임의의 지령들 실례로 시간을 표시하는것과 같은 지령들을 보존하기 위하여 파일 .bash_logout을 리용할수 있다. Korn은 이와 비슷한 파일을 가지지 않는다.

bash와 Korn쉘은 Bourne쉘의 상위모임들이며 bash는 또한 C쉘에서 리용되는 일부 기능들도 가지고 있다(만일 Linux를 사용한다면). 기정적인 가입쉘로서 Korn쉘과 bash를 리용하는것이 적합하다. 이 장을 납득했다면 그것을 설정하는 체계관리자에게 그 방법을 문의하시오.

요 약

어떤 쉘에서 지령은 다른 쉘에서 유효하지 않거나 혹은 다른 방식으로 실행된다. Bourne쉘은 C쉘의 일부 기능들을 가지고 있다. Korn과 bash쉘은 기능상 대단히 풍부하며 리용에 가장 적합하다.

환경변수들은 모든 보조쉘들에서 리용할수 있도록 반출된 쉘변수들이다. UNIX체계의 동작은 이 변수들의 설정에 의하여 크게 좌우된다. 가입쉘과 vi, emacs, telnet에서 탈퇴지령에 의하여 실행되는 쉘은 대화형쉘들이다. 쉘스크립트들은 비대화형쉘과 함께 실행된다. 이 두개의 쉘들은 동작패턴에서 차이가 있다.

매 초기화스크립트는 보조쉘이 없이 실행되며 따라서 스크립트에서 정의된 변수들은 가입쉘과 보조쉘들에서 유효하게 된다. 점(.)지령은 Bourne, Korn, bash에서 이러한 처리를 진행하며 source지령은 C쉘에서 리용된다(이 지령은 bash에서도 리용된다).

Bourne쉘이 아닌 쉘들에서 별명은 alias문으로서 긴 지령렬들을 생략할수 있게 한다. 이 쉘들은 이전 지령들을 다시 호출하는 리력기능을 지원하며 필요하다면 치환도 진행할수 있다. 또한 인수 noclobber와 iguoreof를 리용하여 파일에 대한 우연한 재쓰기와 [Ctrl-d]에 의한 체계의 탈퇴를 막을수 있다. 물결표(~)는 홈등록부의 생략형식으로서 동작한다.

Korn과 bash쉘은 대화형쉘로 그리고 비대화형쉘로도 리용될수 있다.

Bourne쉘(Korn과 bash에도 적용할수 있다.)

환경변수들을 리용하여 지령탐색경로(PATH)를 설정할수 있으며 등록부탐색경로(CDPATH)를 설정할수 있다. 홈등록부와 가입쉘은 파일 /etc/passwd로부터 결정되며 HOME과 SHELL에서 유효하다. 말단이 적당히 동작하도록 하자면 변수 TERM이 정확히 설정되여야 한다. 쉘은 우편함의 위치(MAIL)와 새로운 우편을 찾기 위한 검사간격(MAILCHECK)을 알고 있다.

지령 export는 일반변수들을 환경변수들로 만든다. 환경변수에로의 변경은 사용자가 체계에 가입할 때 쉘에 의하여 실행되는 스크립트 $HOME/.profile에 보존된다.

C쉘

set명령문은 국부변수들에 값을 할당하며 또한 그것들을 표시한다. setenv명령문은 환경변수들을 설정하며 인수없이 리용하면 그 변수들에 대한 설정들을 화면에 표시한다. TERM과 같은 대문자변수들은 프로그람에 의하여 읽혀 지지만 C쉘은 소문자변수들만을 리용한다.

사용자는 또한 지령탐색경로(path)와 프롬프트문자렬(prompt)을 설정할수 있다. 홈등록부는 변수 home에, 현재등록부는 변수 cwd에, 쉘은 변수 shell에 그리고 우편등록부는 변수 mail에 기록된다. 이 많은 특수변수들은 환경변수들로서 대문자변수를 가지고 있다.

별명은 또한 같기기호(=)없이 정의된다. 기호 \!$은 한개의 마지막파라메터만 표현하며 \!*는 모든 파라메터를 나타낸다.

이전 지령들은 기억기(history)에 혹은 파일 .history(saavehist)에 보존된다.!는 문자렬과 함께 리용되면 이전 지령을 실행시키며 !!는 마지막지령을 반복실행한다. !$와 !*는 각각 마지막지령의 마지막인수와 그 지령의 모든 인수들을 표현한다.

:s를 리용하여 이전 지령에서 치환을 진행할수 있다. 기호렬 ^s1^s2은 마지막지령에 대하여 치환을 진행한다.

:n연산자(n은 0으로부터 시작한다.)는 이전 지령의 하나 또는 여러개의 인수들을 선택한다. 사용자

는 다른 인수모임을 가진 이전 지령을 실행시킬수 있다(:0). 경로이름으로부터 머리부(:h)와 꼬리부(:t) 그리고 확장자 없는 기본파일이름(:r)을 취할수 있다.

우연적인 파일재쓰기와 체계탈퇴는 set noclobber와 set ignoreeof에 의하여 보호된다.

파일 ~/.cshrc와 ~/.login들은 사용자가 체계에 가입할 때 실행된다. 파일 .cshrc은 보조쉘을 기동할 때 리용되며 파일 .logout는 사용자가 체계에서 탈퇴할 때 실행된다. 별명들과 특수한 소문자변수들에 대한 설정은 .cshrc파일에 보존되여야 한다.

Korn쉘

Korn쉘은 Bourne쉘에서의 모든 변수들을 사용한다. 사용자는 프롬프트문자렬에 현재등록부(PWD)와 사건번호(!)를 포함시킬수 있다.

별명들은 같기기호(=)로 정의되며 지령행인수들을 받아 들이지 않는다.

이전의 지령들은 파일 ~/.sh_history(혹은 변수 HISTFILE에 정의된 파일) 에 보존된다. 문자렬과 함께 리용된 r는 이전의 지령을 반복한다. 그자체로 리용되면 r는 마지막지령을 반복한다. $_는 마지막지령에 대하여 마지막인수를 표현한다.

vi 혹은 emacs행내편집기능들은 인수 vi 혹은 emacs를 가진 set -o지령에 의하여 가능하게 된다. 사용자는 지령행에 입력된 문자렬을 확장하는 지령 및 파일이름완성하기기능을 리용할수 있다.

set -o지령은 파일에 대한 덧쓰기(noclobber)와 우연한 체계탈퇴(ignoreof)를 막기 위하여 리용된다. 이전 등록부로 바꾸기 위하여 cd -를 리용할수 있다.

파일 .profile에 대하여 추가적으로 본다면 가입쉘과 대화식보조쉘들은 둘 다 변수 ENV에 정의된 환경파일(보통 .kshrc)을 실행한다. 환경변수들과 set -o명령문 그리고 별명들은 환경파일에 보존되여야 한다.

bash

bash는 Bourne쉘의 상위모임이며 bourne쉘의 모든 변수들을 리용한다. 프롬프트문자렬(PSl)은 현재등록부(PWD)와 사건번호(!)를 포함할수 있다. 또한 프롬프트에 사용자이름, 주콤퓨터이름 그리고 현재시간도 포함시킬수 있다.

별명은 같기기호(=)로 정의되며 지령행인수들을 받아 들이지 않는다.

이전의 지령들은 파일 .bash_history(혹은 변수 HISTFILE에 정의된 파일)에 보존된다. !뒤에 문자렬이 오면 부분문자렬로 시작하는 가장 최근의 지령을 실행한다. 또한 !!으로서 마지막지령을 반복할수 있다. $_와 !$이 마지막지령의 마지막인수를 표현한다면 !*는 이전 지령의 모든 인수들을 표현한다.

사용자는 이전 지령에서 치환(:s와:gs)을 진행할수 있다. bash는 또한 마지막지령에서 치환을 진행하기 위한 간략형식의 지령(^s1 ^s2)을 제공한다.

또한 이전 지령에 대하여 하나이상의 인수들을 취할수 있으며(:n) 다른 인수들을 가진 이전 지령을 실행시킬수 있다(:0). 경로이름으로부터 머리부(:h)와 꼬리부(:t) 그리고 확장자가 없는 기본파일이름을 취할수 있다.

사용자는 인수로서 vi 혹은 emacs를 가진 set -o지령을 실행하는것으로써 vi 및 emacs행내편집을 진행할수 있다. 또한 지령행에 입력된 문자렬을 확장하여 주는 지령 및 파일이름완성기능을 리용할수 있다.

지령 set -o noclobber은 파일의 우연적인 덧쓰기를 막기 위한 안전방식이다. 또한 ignoreeof가 설정되면 [Ctrl_d]로써는 체계탈퇴를 할수 없다. 지령 cd -는 이전의 등록부로 이행한다.

사용자가 체계에 가입할 때 bash는 파일 .bash_profile,.profile,.bash_login들중에서 처음으로 발견된 파일을 실행한다. 변수 BASH_ENV은 보조쉘이 기동될 때 실행되는 파일(보통 .bashrc)을 결정한다. 모든 환경변수들, set -o명령문들과 별명들은 이 파일에 보존된다.

시험문제

아래의 질문들은 사용자가 Korn 혹은 bash를 자기의 가입쉘로서 가지고 있다는것을 전제로 한다. 사용자는 문맥으로부터 그것을 추론해야 할것이다.

1. 변수 TERM이 값 vt220을 가지고 있다면 어디에서 그의 조종파일을 찾을수 있겠는가?
2. 쉘은 우편에 대하여 얼마만한 시간간격으로 우편함을 검사하는가?
3. 지령 man이 자기의 페지화프로그람(pager)을 설정하기 위하여 변수를 리용한다면 less로 페지화프로그람을 어떻게 변경시킬수 있겠는가.?
4. 어느 변수가 who am i지령과 대응되는가?
5. >기호는 파일들이 우연적으로 덧쓰기되게 하는 위험성이 있다. 어떤 방책이 있는가?
6. 만일 쉘스크립트를 실행시키는 경우 보조쉘이 가입파일(.login 혹은 .profile)을 읽는가?
7. 어느 때 쉘변수가 환경변수로 되는가?
8. 홈등록부에 파일 .bash_profile 혹은 .profile이 있다면 체계에 가입할 때 bash가 그것들을 둘 다 읽겠는가?
9. C쉘, Korn, bash에서 리력기능이 마지막 200개의 지령들을 기억기에 어떻게 보관하는가?
10. C쉘, Korn, bash에서 마지막지령을 어떻게 반복할것인가?
11. 지령행에서 vi식편집을 가능하게 하자면 Korn쉘, bash에서 먼저 무엇을 설정해야 하는가?
12. 어떤 사용자에 대한 홈등록부의 절대경로이름을 모른다고 하자. Bourne쉘이 아닌 다른 쉘들에서 그곳으로 어떻게 이행하겠는가?
13. 기호 :h와 :t들의 의미는 무엇인가?

련습문제

1. 가입쉘을 변경시키자면 체계관리자와 접촉해야 하는가?
2. 파일 /etc/passwd를 읽어서 설정되는 2개의 환경변수는 어느것인가?
3. 현재 존재하는 PATH변수에 어미등록부를 어떻게 추가하겠는가?
4. 자기의 우편함을 매분마다 검사하자면 어떤 설정을 해야 하는가?
5. bash에서 다음과 같은 프롬프트를 어떻게 만들수 있는가(사용자: romeo, 콤퓨터이름: niccomail, project: 현재등록부)?

   [niccomail-romeo~/project5]
6. 자기가 현재등록부 /home/romeo/cgi에 있다고 가정하자. 지령 cd perl로써 등록부 /home/romeo/perl에로 이행하려면 어떤 설정이 필요한가?

7. PS1='\!$'에 대하여 Korn과 bash에서 어떤 형식의 프롬프트를 보여 주는가?
8. 현재등록부의 숨겨 진 파일들만을 보여 주는 Korn 혹은 bash별명을 만들어 보시오. 만일 숨겨 진 파일이 등록부라면 출력이 어떻게 달라 지는가?
9. (1)현재 등록부에서 실행파일들만을 표시하는 Korn 혹은 bash별명을 만드시오. (2)vi로 편집하기 위하여 마지막으로 변경된 파일을 취하는 Korn과 bash별명을 만드시오.
10. C쉘에서 자기 프롬프트에 현재등록부를 반영하자면 어떤 설정을 해야 하는가(참고 : cd지령에 대한 별명을 생각해 보시오)?
11. 덧쓰기를 막기 위해 noclobber를 설정한 경우 C쉘, Korn과 bash에서 실제로 덧쓰기가 필요하다면 어떻게 해야 하는가?
12. 등록부 /etc에 파일 profile(점이 없다.)이 있다. 이 파일이 실행되겠는가?
13. 만일 파일 .alias에서 모든 별명들을 보존하였다면 C쉘, Korn, bash의 모든 보조쉘들에서 그것들이 유효하다는것을 어떻게 확인할수 있는가?
14. C쉘, Korn, bash에서 몇개의 건누름으로써 마지막 fdformat지령을 어떻게 빨리 재실행시킬수 있는가?
15. 아래의 지령은 무엇을 의미하는가?

```
^htm^df
```

16. 자기가 방금 지령 tar -cvf /dev/fd0 *.sh를 리용하였다고 하자. C쉘, Korn, bash에서 .pl파일들을 리용한 우와 같은 지령을 어떻게 반복실행하겠는가?
17. 지령 $_를 실행시켜 통보 foo.sh:Permission denied를 받았다고 하자. 이것은 무엇을 가리키는가.
18. 직렬지령편집이 가능하다고 가정하면 vi방식으로, emacs방식으로 편집하기 위하여 이름 perl을 포함하는 마지막지령을 어떻게 불러 들이겠는가?
19. Korn쉘, bash에서 PATH에 의하여 지적된 등록부들에서 z로 시작하는 모든 지령이름들을 보기 위한 가장 쉬운 방법은 무엇인가?
20. C쉘, Korn, bash에서 다음의 렬을 생략할수 있는가?

```
vi script.c:cc script.c
```

21. C쉘, Korn, bash에서 자기가 마지막으로 보았던 등록부로 어떻게 옮겨 갈수 있겠는가?
22. 대화형쉘과 비대화형쉘사이에 어떤 차이가 있는가? 쉘의 어떤 기능들이 대화형쉘에서만 의미를 가지는가?
23. 지령 . .profile이 사용될 때 체계는 오유통보문 ksh:.prcfile:not found를 발생시킨다. 현재등록부에 .profile이 존재하는데도 불구하고 왜 이러한 통보문이 발생하는가?
24. 만일 .profile파일이 PATH=$PATH:$HOME/bin과 같은 명령문들을 포함하고 있고 그 파일에 대한 변경을 반복하였다면 이러한 변경을 능동으로 하기 위해서는 어떻게 해야 하는가?
25. 명령문 vi !un:l:r의 의미는 무엇인가?

제 18 장. 쉘프로그람작성

쉘의 동작은 지령해석 하나만으로 제한되지 않는다. 즉 그보다 훨씬 더 많은것을 포함한다. 쉘은 변수, 조건식, 순환명령을 가진 어떤 언어와 같이 하나로 결합된 내부지령전체의 모임이다. 대부분의 구조들은 C형식을 모방하였으며 사용에 있어서 C보다 더 치밀하고 간결하다. 쉘프로그람들의 성능이 높은 원인은 UNIX의 외부지령들이 쉘의 내부구조들과 쉽게 어울릴수 있기때문이다.

쉘프로그람작성에 대해서는 2개의 장으로 나누어 취급한다. 이 장에서는 쉘프로그람작성기능에 대하여 론의하며 모든 쉘의 《최소공통분모》인 Bourne쉘에 초점을 둔다. 다른 장에서는 Korn과 bash의 개선된 기능들을 론의한다. 그러나 여기서 론의하는 모든것들은 이 두개의 쉘들에도 적용할수 있다. C쉘은 전혀 다른 프로그람작성구조를 리용하며 부록 1에서 개별적으로 취급한다.

쉘프로그람은 한번에 한개 명령문씩 해석방식으로 실행된다. 따라서 고수준언어로 씌여 진 프로그람보다 실행속도가 뜨다. 그러나 이것은 일감을 처리하는데서 주요한 인자로 되지는 않는다. 많은 경우에 쉘은 특히 체계관리자과제들에서 대단히 우월하다. UNIX체계관리자는 완전한 쉘프로그람작성자이여야 한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 쉘스크립트를 실행하는 여러가지 방법에 대하여 배운다(18.2).
- read지령으로 건반과 파일로부터 자료를 얻는 방법에 대하여 배운다(18.3).
- 스크립트내부에서 지령행인수들이 위치파라메터들로서 어떻게 보내지며 해석되는가 하는것을 본다(18.4).
- 지령의 완료값의 의미와 파라메터 $?의 의미를 파악한다(18.5).
- 연산자 &&와 ||로서 간단한 한행조건식을 리용하는 방법을 배운다(18.6).
- UNIX지령을 여러 갈래로 분기하여 실행시키는 if명령문에 대하여 배운다(18.8).
- 문자렬 및 옹근수를 비교하고 test지령으로써 파일속성을 시험한다(18.9).
- 통용기호로 문자렬을 정합하기 위한 case의 패턴정합기능에 대하여 배운다(18.10).
- expr로 옹근수값계산 및 문자렬처리를 진행한다(18.11).
- 호출되는 이름에 따라 다르게 동작하는 스크립트설계방법을 배운다(18.12).
- while과 until순환을 리용하여 지령묶음을 반복적으로 실행시킨다(18.14).
- for순환을 리용하여 목록의 매 성원들에 대한 명령묶음을 실행시킨다(18.16).
- basename으로 파일이름확장자를 변경시킨다(18.16).

주해

여기서 취급되는 모든것들은 Bourne쉘은 물론 Korn과 bash쉘에서도 적용된다. 그러나 C쉘에는 완전히 적용되지 않는다. bash를 리용할 때 관심을 돌려야 할것이 있다. 하나는 확장문자렬 \c와 \n을 사용할 때마다 echo -e를 리용해야 한다는것이고 다른 하나는 특수변수 $0이 Bourne과 Korn쉘에서 서로 다른 의미를 가진다는것이다. 이러한 차이점에 대해서는 다음장에서 지적한다.

18.1 쉘변수

값을 기억시키는 편리한 방법으로서의 쉘변수들에 대해서는 제8장에서 소개되였다. 이 장에서는 이러한 변수들을 쉘스크립트들에서 리용할것이다. 간단히 요약하면 변수이름은 문자로서 시작해야 하며 수자와 밑선기호를 포함할수 있다. 특히 변수의 값주기에 대해서는 그앞에 기호 $를 리용하지 않으며 같기기호(=)에 의하여 값이 할당된다. 실지로 $는 변수의 값을 의미한다.

```
$ fname=profile
$ echo $fname
profile
```

unset명령은 쉘로부터 변수를 제거한다. 변수들은 옆으로 차례로 놓는것으로써 련결된다. 즉 다른 프로그람작성언어에서와는 달리 연산자가 필요 없다.

```
$ x=foo ; y=.doc            한개 행에서 여러개의 값주기를 진행할수 있다
$ z=$x$y                    변수 z는 다른 변수들에 의하여 값이 할당된다
$ echo $z
foo.doc
```

쉘은 변수의 값을 표현하기 위한 표식기호로서 변수이름을 둘러 싸는 괄호 {와 }를 리용한다. 아래의 지령에서는 변수의 값을 표현하기 위한 또 다른 방법을 리용하였다.

```
$ echo ${fname}
profile
```

쉘은 다양한 처리를 진행하기 위하여 우와 같은 형식을 요구한다. 변수와 문자렬은 두개의 변수를 련결할 때와 같은 방법으로 서로 련결할수 없다. 실례로 변수의 값에 문자 x를 첨부하기 위해서는 다음의 두가지 형식중에서 어느 하나를 사용해야 한다.

```
$ echo ${fname}x
profilex
$ echo $fname"x"            외인용부호도 쓸수 있다
profilex
```

여기서 우리는 파일확장자를 변경할 때 이러한 련결기능이 쓸모 있다는것을 알수 있다.

지령과 쉘변수들의 협조적인 작용은 명백하고 치밀한 스크립트를 만드는데 도움을 준다. 괄호를 리용하여 3.7에서의 여러개의 지령렬을 아래와 같이 다시 쓸수 있다.

```
boldon=`tput smso`; boldoff=`tput rmso`
echo ${boldon}Come to the Web$boldoff
```

이것은 Come to the Web라는 단어들을 말단에 굵은 글자체로 표시한다. 괄호는 첫번째 변수에 대해서만 요구되며 두번째 변수에는 필요 없다. 우의 실례에서 tput지령렬은 변수들에 설정하였으며 후에 자기의 쉘스크립트에서 직접 리용할수 있다.

18.2 쉘스크립트

리론적으로는 모든 쉘명령문들과 UNIX지령들이 지령행 그자체에 입력된다. 그러나 지령묶음이 규칙적으로 실행되여야 하는 경우에는 파일에 보존하는것이 더 좋다. 이와 같은 파일들을 **쉘스크립트** 또는 쉘프로그람이라고 부른다. 이 파일이름의 확장자에 대해서는 보통 .sh를 리용한다. 그러나 이것이 절대적인것은 아니다. .sh확장자를 리용한다면 통용기호로 스크립트들를 탐색하기가 쉽다.

아래에 보여 주는 쉘스크립트 script.sh는 5개의 UNIX지령들로 이루어 졌으며 그중 네개는 echo지령이다.

```
$ cat script.sh
# Sample shell script
echo "The date today is `date`"               #지령치환의 리용
echo Your shell is $SHELL
echo Your home directory is $HOME
echo The processes running on your system are shown below:
ps
```

사용자는 이러한 스크립트를 만들기 위하여 vi와 emacs편집기를 리용할수 있다. 설명문자 #는 임의의 위치에 놓을수 있으며 쉘은 그 기호의 오른쪽에 놓인 모든 문자들을 무시한다.

사용자는 두가지 방법으로 이 파일을 실행시킬수 있다. 일반적인 방법은 그것을 기동시키기전에 먼저 지령 chmod로써 실행가능하게 만들어야 한다.

```
$ chmod +x script.sh ; ls -l script.sh
-rwxr-xr-x    1 romeo     dialout       154 Feb 25 22:33 script.sh
```

스크립트에 대하여 실행허가가 되여 있으므로 파일이름만을 간단히 입력하여 그것을 실행시킬수 있다.

```
$ script.sh
The date today is Fri Feb 25 22:35:49 IST 2000
Your shell is /bin/sh
Your home directory is /home/reomeo
The processes running on your system are shown below:
PID TTY STAT TIME COMMAND
192   1 S    0:00 sh
588   1 R    0:00 ps
```

모든 명령문들은 련속적으로 실행되였다. 이 스크립트는 대단히 간단하다. 즉 그 어떤 입력도 없고 지령행인수도 없으며 조종구조도 없다. 앞으로 스크립트들에 이 기능들을 점차적으로 추가할것이다.

쉘스크립트는 항상 개별적인 쉘 즉 보조쉘들에서 실행된다. 스크립트가 실행될 때 ps지령은 적어도 두개의 sh(다른 쉘들에 대하여서는 ksh와 bash)프로쎄스들이 실행되고 있다는것을 보여 준다. 가입쉘은 실제로 스크립트를 실행시키는 보조쉘의 어미이다. 스크립트에 while과 같은 순환명령이 있으면 몇개의

보조쉘들이 더 만들어 진다. 즉 스크립트가 실행상태에 있을 때에는 적어도 두개의 프로쎄스들이 실행되고 있다는것을 알아야 한다.

그러면 스크립트를 실행시키기 위해서는 어느 쉘을 리용해야 하겠는가? 사용자는 본능적으로 가입쉘을 리용하려 할것이다. 그러면 현재의 가입쉘이 Korn인 조건에서 Bourne쉘에서 씌여 진 스크립트를 가지고 있다면 어떻게 되겠는가? 사용자는 아래와 같이 그것을 실행시키려 할것이다.

```
script.sh                혹은 ./script .sh
```

이렇게 하면 스크립트를 실행시키기 위하여 Korn보조쉘을 호출할것이다. 아마도 이것은 오유를 발생시킬것이다. Bourne쉘은 모든 체계에서 일반적으로 유효하므로 사용자는 자기의 보조쉘로써 그것을 호출할수 있다.

```
sh script.sh                보조쉘을 명백히 요구한다
```

여기서는 인수로서 스크립트를 가진 sh지령을 실행시키고 있다. sh지령은 스크립트파일을 열고 그안의 모든 명령문들을 련속적으로 실행시킨다. 스크립트파일에 대한 실행허가는 필요 없다.

주해

사용자는 현재등록부가 변수 PATH에 반영될 때에만 파일이름을 호출하는것으로써 스크립트 foo를 실행시킬수 있다. 경우에 따라 그렇지 않을수도 있다. 사용자는 그러한 경우에도 절대경로이름을 쓰지 않고 스크립트를 실행시킬수 있다. ./foo를 리용하시오.

만일 자기의 쉘스크립트 혹은 perl스크립트를 편집하기 위하여 vi편집기를 리용한다면 그 스크립트를 실행시키기 위하여 편집기에서 탈퇴하지 않아도 된다. 이 경우 파일 $HOME/.exrc에 [f1]기능건에 대한 대응관계를 표현하는 기호렬을 작성하시오.

```
: map #1 ^[!w^M!!%^M
```

참고

이것은(:!%) 파일을 실행시키기전에 완충기에 보존하며 ^[와 ^M은 각각 [ESC]와 [Enter]건을 표현한다. 사용자는 이 상태에서 현재스크립트를 실행하기 위하여 F1건을 누를수 있다(그러나 먼저 지령 chmod으로서 스크립트를 실행가능상태로 만들어야 한다). 만일 스크립트가 인수들을 받아 들인다면 ^M을 제거한 나머지부분을 [F2]건에 대응시키시오. 다음 [F2]기능건을 누른후에 인수들을 입력하시오.

해석기의 지정

스크립트내부에서 쉘을 지정하는것도 가능하다. 그러므로 《sh 파일이름》으로써 스크립트를 실행키는것보다 스크립트의 시작부분에 아래의 명령문을 놓는것이 더 좋다.

```
#!/bin/sh                sh는 /bin안에 있다고 가정한다
```

#는 쉘에서 설명문자로서 취급되지만 그뒤에 !를 붙이면 쉘은 해석기서술로서 그 나머지문자렬을 해석한다. 이렇게 하면 sh를 명백히 호출하지 않고도 스크립트를 실행시킬수 있다. 만일 스크립트를 어떤 특정한 쉘하에서만 실행시키고 싶다면 스크립트의 제일 웃부분에 우와 같이 해석기에 대한 서술을 하시오(그전에 아무것도 기입해서는 안된다).

주해

일부 UNIX체계들에서 C쉘은 스크립트실행을 위한 쉘선택에 따라 다르게 동작한다. 이러한 체계들에서 가입쉘이 csh이라면 시작부분에 다음명령을 써넣을 필요가 있다.

```
#!/bin/csh
```

즉 안전하게 동작하게 하자면 모든 쉘스크립트의 제일 웃부분에 해석기에 대한 서술을 진행해야 한다.

18.3 스크립트를 대화식으로 작성하기(read)

read명령문은 사용자로부터 입력자료를 얻기 위한 쉘의 내부도구로서 스크립트를 대화식으로 작성할수 있게 한다. 이 명령문은 하나이상의 변수와 함께 리용된다. 표준입력장치를 통하여 제공되는 입력자료는 이 변수들에 반영된다.

```
read name
```

우의 지령을 리용하면 스크립트는 건반으로부터 자료를 얻기 위하여 그 시점에서 실행이 중지된다. 이것은 변수의 값주기형식이기때문에 name앞에 기호 $를 리용하지 않았다. 아래에 제시된 스크립트 empl.sh는 말단으로부터 탐색문자렬과 파일이름을 얻기 위하여 read명령문을 리용한다.

```
$ cat empl.sh
#!/bin/sh
# Script: empl.sh - Interactive version
# The pattern and filename to be supplied by the user
echo "Enter the pattern to be searched: \c" #\n for a newline
read pname
echo "Enter the file to be used:  \c"
read pname
echo "Searching for $pname from file $flname"
grep "$pname" $flname
echo "Selected lines shown above"
```

쉘스크립트는 행의 임의의 위치에 설명문들을 가질수 있다. 이때 설명문앞에 기호 #를 붙여야 한다. 우의 프로그람에서 확장문자렬 \c이 무엇을 하는지 알고 있으리라고 생각한다(8.5). 스크립트를 실행시키고 스크립트가 두번 정지할 때마다 입력자료를 넣으시오.

```
$ empl.sh
Enter the pattern to be searched: director
Enter the file to be sued: emp2.lst
Searching for director from file emp2.lst
9876|bill johnson  |director  |production|03/12/50|130000
2365|john woodcock |director  |personnel |05/11/47|120000
Selected lines shown above
```

먼저 스크립트는 입력될 패턴에 대해 문의한다. 문자렬 director를 입력하시오. 쉘은 변수 pname에 그 문자렬을 할당한다. 다음 스크립트는 파일이름에 대하여 문의한다. 문자렬 emp2.lst를 입력하시오. 쉘은 변수 flname에 그 값을 보존한다. 건반으로부터 이러한 자료들을 받아 들인후에 grep와 echo지령들이 실행된다.

[Enter]건를 누르기전에 같은 행에 두개의 인수들을 다 입력할수 있지만 그렇게 하자면 두개의 변수를 가진 read지령을 리용해야 한다.

```
echo "Enter the pattern and filename: \c"
read pname flname
```

주해

입력된 인수들의 개수가 그것을 받는 변수의 개수보다 작으면 나머지변수들은 대입되지 않은 채로 남아 있게 된다. 그러나 인수의 개수가 변수의 개수를 초과하는 경우에는 남은 단어들은 마지막변수에 대입된다.

read입력의 방향절환

우에서 설명한 스크립트는 대화식스크립트이다. 즉 그의 인수들은 read지령에 의하여 표준입력장치로부터 읽혀 진다. 그러면 방향절환을 진행하여 스크립트가 파일로부터 자료를 얻게 할수 없겠는가? 파일 list에 다음과 같은 2개의 행을 삽입하자.

```
$ cat list
director
emp2.lst
```

다음 스크립트 empl.sh를 실행시키고 이때 자료를 이 파일로부터 취하도록 방향절환시킨다.

```
$ empl.sh<list
Enter the pattern to be searched: Enter the file to be used:
Searching for director from file emp2.lst
9876|bill johnson  |director   |production|03/12/50|130000
2365|john woodcock |director   |personnel |05/11/47|120000
Selected lines shown above
```

스크립트는 모두 두개의 프롬프트를 보여 주지만 그때마다 정지되지는 않는다. 즉 대화식스크립트를 비대화식으로 실행시킬수 있다는것을 보여 준다. 이 기술은 보통 고정된 형식의 응답문들을 가지고 리용되는 안내형식의 구동형프로그람을 작성하는데 쓸모가 있다. 이와 같이 사용자는 건반이 아니라 파일로부터 이러한 응답들을 읽게 할수 있다. 그렇다면 파일로부터 vi지령들을 취할수 없겠는가? 표준입력장치를 통한 vi의 명령들에 대해서는 뒤부분에 가서 구체적으로 론의한다.

18.4 위치파라메터

쉘스크립트들은 지령행 그자체로부터 인수들을 받아 들인다. C와 perl언어들도 이 방법으로 인수들을 사용한다. 사실상 이것은 일반적으로(C로 서술된) UNIX도구들을 리용하는 방법이다. 지령행인수들을 지적하는 이러한 비대화식방법은 방향절환, 관흐름과 함께 리용되는 도구들을 개발하기 위한 기초로 된다.

인수들이 쉘스크립트와 함께 지적되면 그것들은 이미 정해 진 특수변수 즉 **위치파라메터**(positional parameter)들에 할당된다. 다시말하여 인수들은 쉘에 의하여 파라메터 $1, $2,...에 보존된다. 모든 변수들은 자기 이름앞에 $을 붙이는것으로써 그 값이 평가되며 그렇다고 하여 우에서 지적된 파라메터 $1, $2,...을 쉘변수라고 부를수는 없다. $1의 경우 실제로 $로 평가되는 변수 1이 존재하는것이 아니다.

그러면 지령행인수들을 받기 위하여 이전의 스크립트를 다시 써보자. 여기서는 스크립트안에 우와 같은 특수파라메터들을 사용한다.

$1 -첫번째 인수

$2 -두번째 인수

$0 -스크립트의 이름

$# -인수의 개수

$* -단일문자렬로서 위치파라메터들에 대한 완전한 묶음

인수들을 표현하기 위한 파라메터로서 추가적으로 특수파라메터 $0, $#, $*를 리용할수 있다. 이 장에서는 이러한 파라메터들이 여러번 사용될것이다. 우선 스크립트 empl.sh에서 "passive"방식으로 그것들을 사용하여 보자.

```
$ cat emp2.sh
#!/bin/sh
echo "Program: $0"                # $0은 프로그람이름을 포함한다
echo "The number of arguments specified is $#"
echo "The arguments are $*"                # 모든 인수들은 $*에 저장된다
grep "$1" $2
echo "\nJob Over"
```

7.13에서 론의한 련결에 대하여 다시 상기해 보자. 거기서 우리는 프로그람을 여러가지 이름으로 호출하는데 대하여 론의하였다. $0은 호출되는 스크립트의 이름을 가리킨다. 만일 쉘이 $#으로써 인수의 개수를 계수한다면 사용자는 정확한 개수의 인수가 입력되였는가를 검사하기 위한 스크립트를 설계할수 있다.

패턴 director와 파일이름 emp.lst를 두개의 인수로 하는 다음의 스크립트를 보자.

```
$ emp2.sh director emp1.lst
Program: emp2.sh
The number of arguments specified is 2
The arguments are director emp1.lst
1006|gordon lightfoot|director |sales      |09/03/38|140000
6521|derryk o'brien  |director |marketing  |09/26/45|125000


Job Over
```

인수들이 이런 방법으로 정의될 때 첫 단어(지령 그자체)는 $0에, 두번째 단어(첫번째 인수)는 $1에 그리고 세번째 단어(두번째 인수)는 $2에 할당된다. 이런 방법으로 위치파라메터들을 $9까지 사용할수

있다(shift명령문을 리용하면 그이상 더 쓸수 있다).

우의 스크립트는 두개의 인수들을 리용한다. 그러면 robert bylan와 같이 여러 단어로 이루어 진 패턴에 대한 탐색은 어떻게 하겠는가? 문자렬이 인용부호로 둘러 막히면 쉘은 그것을 하나의 인수로서 인식한다.

```
$ emp2.sh "robert dylan" emp1.lst
Program: emp2.sh
The number of arguments specified is 2
The arguments are robert dylan empl.lst
5678|robert dylan  |d.g.m.    |marketing |04/19/43| 85000

Job Over
```

$#는 여전히 2로 설정된다. 만일 인용부호를 리용하지 않았다면 인수의 개수는 3으로 되며 grep지령은 파일이름을 bylan으로 해석하였을것이다. 이것은 물론 실패이다.

주해

$0을 리용하면 쉘스크립트는 자기자체의 이름을 인식한다. 만일 스크립트의 파일이름이 련결을 가지고 있다면 사용자는 호출하는 이름에 따라 여러가지로 동작하게 할수 있다. 우리는 case명령문에 대한 사용법을 배운후에만 이 기능의 편리한 점을 알게 될것이다.

BASH쉘

bash에서 $0은 약간 다르게 설정된다. 즉 우의 실례에 대하여 $0은 emp2.sh가 아니라 ./emp2.sh로 설정될것이다. 후에 보게 되는 스크립트들은 $0의 값을 파일이름과 비교할것이다. 이 경우 비교를 진행하기 위해 ./을 제거해야 한다. 물론 이에 대해서는 배우게 될것이다.

18.5 지령의 완료상태

grep지령이 패턴을 찾아 내지 못했다면 우리는 지령이 실패하였다고 보았다. 지령은 실패할수도 있으며 사용자는 그 결과에 대하여 명백히 알아 보아야 한다. 매장들에서 UNIX지령들을 리용할 때 일부 지령들은 자기가 원하는대로 실행되였지만 그렇지 못한것들도 있었다. 실례로 지령

```
$ cat foo
cat: can't open foo
```

은 실패한다. 즉 표준오유출력장치에 파일이 없거나 혹은 그 파일을 읽을수 없다는 오유통보문이 표시된다. 우리는 이미 grep가 패턴탐색에서 실패하는 경우 간단히 프롬프트상태를 돌려 준다는데 대하여 이미 알고 있다.

모든 지령은 실행이 끝나면 값을 되돌려 준다. 이 값을 지령의 **완료값**(exit status) 또는 **돌림값**(return value)이라고 부른다. 그 값은 지령이 성공적으로 실행되면 참으로 되고 실패하면 거짓으로 된다. 우에서 본 cat지령은 실패하였으며 완료값이 거짓이라는것을 알수 있다.

이 돌림값은 프로그람작성자들에게 있어서 대단히 중요한것이다. 프로그람작성자들은 지령의 성공혹은 실패에 따라 서로 다른 실행경로를 가지는 프로그람론리를 구성하는데 이 값을 리용한다. 실례로 중요한 파일이 없거나 읽혀 지지 않는 경우 스크립트실행은 중지되지 않는다. 쉘은 돌림값을 검사하는 명령문(test)을 제공한다.

파라메터 $?

쉘에 의하여 리용되는 또 하나의 특수파라메터 $?가 있다. 이것은 마지막지령의 완료값를 보존한다. 지령이 성공하면 그 값은 0으로 되며 실패하면 0이 아닌 값으로 된다. 실례로 grep가 패턴탐색에서 실패하면 돌림값은 1로 되며 탐색된 파일이 첫번째 위치에서 읽혀 지지 않는다면 돌림값은 2로 된다. 어떤 경우에나 0이상의 돌림값은 지령의 실패로서 해석된다. 그러면 여러가지 방법으로서 grep지령을 리용해 보자.

```
$ grep director emp.lst>/dev/null; echo $?
0
$ grep manager emp.lst>/dev/null; echo $?
1
$ grep manager emp3.lst>/dev/null; echo $?
grep: can't open emp3.lst
2
```

위치파라메터와 특수파라메터들은 쉘에 의하여 자동적으로 할당된다. 사용자들은 간접적인 방법을 제외하고는 파라메터들의 값들을 임의로 변경할수 없다. 그러나 여러가지 방식으로 그것들을 아주 편리하게 리용할수 있다. 이것들은 쉘스크립트들에서 여러번 반복되여 리용될것이다. 이 파라메터들을 표 18-1에 보여 주었다. 여기서 두개는 이미 앞에서 론의된것이며 하나는 후에 론의할것이다.

주해

지령이 성공적으로 실행되였는가 그렇지 않은가 하는것을 알자면 지령다음에 echo $?를 간단히 실행시키시오. 0은 성공을 가리키며 그외의 값들은 실패를 지적한다.

표 18-1. 쉘에서 리용되는 특수한 파라메터

쉘파라메터	의 미
$1,$2,등	위치파라메터
$*	단일문자렬로서 위치파라메터의 완전한 묶음
$#	지령행에 지적된 인수의 수
$0	실행되는 지령의 이름
$@	겹인용부호로 둘러 막힌 경우를 제외하고 $*와 같다.
$?	마지막지령의 탈퇴상태
$$	현재쉘의 PID(10.3)
$!	마지막배경일감의 PID(10.10.2)

18.6 조건부실행을 위한 론리연산자(&&와 ||)

스크립트 empl.sh에는 패턴탐색이 실패하는 경우 selected lines shown above라는 통보문을 표시하는 론리가 없다. 그것은 grep의 완료값을 프로그람의 흐름을 조종하는데 리용하지 않았기때문이다. 쉘은 조건실행을 위한 두개의 연산자 &&와 ||를 가지고 있다. 이 연산자들의 문법적형태는 다음과 같다.

```
cmdl && cmd2
cmdl || cmd2
```

두 지령은 &&에 의하여 구분되며 지령 cmd2는 cmd1이 성공하는 경우에만 실행된다. 이 연산자를 다음의 방법으로 지령 grep와 함께 리용해 보자.

```
$ grep 'director' empl.lst && echo "pattern found in file"
1006|gordon lightfoot  |director  |sales     |09/03/38|140000
6521|derryk o'brien    |director  |marketing |09/26/45|125000
pattern found in file
```

||연산자는 반대의 기능을 수행한다. 즉 첫번째 지령이 실패하는 경우에만 두번째 지령이 실행된다. 이 연산자를 리용하여 grep에 의한 패턴탐색이 성공하지 못한 경우 사용자는 실패통보문을 내보낼수 있다.

```
$ grep 'manager' emp2.lst || echo "Pattern not found"
Pattern not found
```

다음장에서 쉘스크립트들을 개발하는데 이러한 연산들을 여러번 리용할것이다. awk(16.4)와 perl(20.7)은 이 연산자들을 리용한다.

18.7 스크립트의 완료(exit)

일부 경우 조건을 검사하여야만 지령이 실패라는것을 알게 된다. 일반적으로 프로그람작성자들은 기본적인 자원이 없는 경우 례를 들어 찾으려는 파일이 없는 경우에도 실행은 중지되지 않으므로 프로그람을 강제로 완료하려 할것이다. exit문은 프로그람을 완료하기 위하여 리용된다. 스크립트에서는 이 명령문이 나타나면 실행이 정지되고 조종권이 이 스크립트를 호출한 프로그람(대체로 쉘)에 돌려 진다. 우리는 첫장에서 가입대화를 완료하기 위하여 이 지령을 리용하였다. 쉘은 스크립트실행의 끝위치를 알고 있기때문에 매 쉘스크립트들의 마지막에 exit문을 놓을 필요는 없다. 이 명령문은 흔히 실패할 위험성을 가진 지령과 함께 리용된다. 이제 패턴탐색이 성공하면 그 결과를 보여 주고 그렇지 않고 어떤 원인으로 탐색이 실패하는 경우에는 실행을 완료하도록 이전 프로그람(emp2.sh)을 수정하여 보자.

```
$ cat emp2a.sh
#!/bin/sh
echo "Program: $0"               #$0은 프로그람이름을 포함한다
echo "The number of arguments specified is $#"
echo "The arguments are $*"                #모든 인수들은 $*에 기억된다
grep "$1" $2>patlist 2>/dev/null  || exit 2              #인수를 가진 exit
```

```
echo "Pattern found -- Contents shown below"
cat patlist
```

프로그람은 grep지령이 실패하면 완료되지만 성공하면 그 내용이 현시된다. 또한 오유들은 /dev/null로 보내진다. 그러므로 $2에 의하여 표현되는 파일이 없다 하여도 grep의 오유통보문은 화면에 현시되지 않는다. exit의 인수에 대해서는 곧 취급된다. 현재에는 스크립트를 전혀 존재하지 않는 파일과 함께 실행시켜 보자.

```
$ emp2a.sh manager emp3.lst
Program: emp2a.sh
The number of arguments specified is 2
The arguments are manager emp3.lst
```

마지막 echo지령은 실행되지 않았으며 patlist의 내용도 표시되지 않았다. 이것은 스크립트가 도중에 완료되였다는것을 보여 준다. 그러면 그의 돌림값은 무엇이겠는가? $?의 값을 검사하시오.

```
$ echo $?
2
```

이것은 스크립트에서 exit의 인수로서 지적된 값이였다. 그 인수는 임의의 값을 가질수 있으며 스크립트는 그러한 시점에서 완료될 때에만 지적된 값을 돌려 준다. 만일 돌림값을 지적하지 않으면 $?는 령(0)을 보여 줄것이다. 그러므로 사용자는 오유발생에 관계없이 참값을 귀환할수 있다. 다시말하여 사용자는 이 돌림값을 조종할수 있다.

이런 방법으로 론리연산자 &&와 !!를 리용함으로써 우리는 쉘의 조건문들을 보았다. 이 연산자들은 일정한 제한이 있으며 간단한 조건판단에 리용된다. 복잡한 조건에 대해서는 if문을 리용해야 한다.

18.8 if문

if문은 조건에 따라 두가지 분기를 가진다. 쉘에서는 다른 언어들에서와 같이 다음의 형식으로서 이 명령문이 리용된다.

```
if  지령이 성공이라면          if  지령이 성공이라면          if  지령이 성공이라면
then                          then                          then
    지령들을 실행시킨다            지령들을 실행시킨다            지령들을 실행시킨다
else                          fi                            elif 지령이 성공이라면
    지령들을 실행시킨다                                          then...
fi                                                          else...
                                                            fi
        형식1                          형식2                          형식3
```

BASIC에서와 같이 if는 then을 요구한다. 이 명령문은 《지령행》에 지적된 지령의 성공 혹은 실패를 평가한다. 지령이 성공하면 그뒤에 오는 지령들이 실행되며 반대로 실패하면 else문이 실행된다. 이 명령문은 형식 2에서 보여 주는것과 같이 항상 필요되지 않는다. if는 자기와 대응하는 fi로서 닫겨 져야

하며 그렇지 않으면 오유가 발생한다.

쉘프로그람작성이 대단히 성능이 높은것은 지령의 성공이 어떤 UNIX프로그람의 돌림값에 의하여 결정된다는것이다. 모든 지령들은 cat와 grep지령에서 본것처럼 값을 돌려 준다.

다음의 실례에서는 먼저 grep지령이 실행되며 if명령문은 프로그람흐름을 조종하기 위하여 그의 돌림값을 리용한다.

```
  $ cat emp3.sh
  if grep "^$1" /etc/passwd 2>/dev/null    #Search username at beginning of line
then
echo "Pattern found - Job Over"
      else
    echo "Pattern not found"
fi
```

이것은 if문이 grep의 돌림값을 검사하는 간단한 if-else구조이다. 스크립트의 나머지부분은 명백하다. 우의 프로그람은 아래와 같이 사용자방화벽의 존재에 대하여 파일/etc/passwd를 탐색할것이다.

```
$ emp3.sh firewall
firewall:x:41:31:firewall account:/tmp:/bin/false
Pattern found - Job Over
```

이것은 홈등록부로서 /tmp와 《쉘》로서 /bin/false를 가진 례외적인 사용자등록자리이지만 그에 대해서는 걱정할 필요가 없다. 사용자는 파일에 존재하지 않는 패턴을 입력하는것으로써 else지령의 기능을 검사할수 있다.

```
$ emp3.sh mail
Pattern not found
```

그러면 왜 패턴탐색에 grep를 리용하였겠는가? 또 특수한 마당에서 패턴을 비교하려는 경우에 awk를 사용할수 없겠는가? grep지령을 awk로 바꾸는것으로써 스크립트를 변경시키자.

```
$ cat emp3a.sh
#!/bin/sh
if awk -F:'$1 -'/$1/'/etc/passwd 2>/dev/null ; then
    echo "Pattern found - Job Over"
else
    echo "Pattern not found"
fi
```

우의 스크립트에서 파라메터 $1이 두번 나타난다는것을 주의하시오. 첫번째것은 awk의 마당식별자이며 두번째는 스크립트의 첫번째 인수를 나타낸다. 스크립트인수들은 awk에 의하여 정확히 해석되도록 외인용부호로서 둘러 막아야 한다. 스크립트에서 then행의 위치를 변경시켰다. 즉 사용자는 ;으로 경계를 구분한 다음 if문과 같은 행에 then문을 놓을수 있다. 그러면 이미 존재하는 사용자ID와 함께 스크립

트를 실행시키자.

```
$ emp3a.sh firewall
firewall:x:41:31:firewall account:/tmp:/bin/false
Pattern found - Job Over
```

이것은 문제가 없으며 바라던바 그대로이다. 그러나 아래의 스크립트실행은 그렇지 않다.

```
$ emp3a.sh mail
Pattern found - Job Over
```

문자렬 mail은 존재하지 않으며 awk지령은 실패하였다. 그렇다고 하여 거짓완료값이 귀환되지는 않았다. if는 오유조건을 찾지 못하였으며 우리는 이에 대하여 잘못 생각하였다. 그러면 오유조건을 어떻게 발생시키겠는가? 사용자는 awk의 출력에서 문자들의 개수를 계수할수 있으며 awk가 실제로 출력자료를 산생시켰는가 하는것을 wc로서 검사할수 있다.

```
if awk -F: '$1 ~'/$1/' { print }' /etc/passwd | wc ; then
```

문제는 이 경우에도 wc가 참을 돌려 준다는것이다. 사용자는 그것이 실제로 령인가 아닌가 하는것을 보기 위하여 wc의 문자계수값을 검사할수 있다. 현재는 이 문제에 대하여 취급하지 않는다. 그러나 앞으로 가면서 즉 수값처리방법에 대하여 배운후에 이 문제를 취급한다.

if-elif에 의한 다중분기

앞에서 우리는 if조건문의 두가지 형태 즉 if-then-fi와 if-then-else-fi을 보았다. 그밖의 세번째 형태 즉 if -then-elif-then-else-fi도 있다. 이 형태에서 else문이 선택적으로 존재한다면 elif문은 필요한것만큼 가질수 있다. 그러면 사용자가 작성한 지령에 대하여 henry, romeo, juliet의 crontab파일들을 검사하는데 이 세번째 형식을 리용해 보자.

```
$ cat cronfind.sh
#!/bin/sh
crondir=/var/spool/cron/crontabs
message="has scheduled the $1 command"
if grep "$1" $crondir/henry ; then
   echo "henry $message"
elif grep "$1" $crondir/romeo ; then
   echo "romeo $message"
elif grep "$1" $crondir/juliet ; then
   echo "juliet $message"
else
   echo "None of the users is using the $1 command
fi
```

여기서 우리는 쉘변수들을 적당히 그리고 대단히 효과적으로 리용하였다. 즉 코드량을 줄이기 위하

여 crontab등록부이름과 통보문의 기본부분을 변수에 따로 보관하였으며 각각 $crondir과 $message로서 그것들을 참조하였다.

세명의 사용자중 누가 crontab파일들에 지령 fine /(뿌리등록부로부터 탐색)을 작성해 넣었는가 하는것을 탐색하기 위하여 이 스크립트를 리용할수 있다.

```
$ cronfind.sh "find */"
59 16 * * * find      / -name "*.html" - print |mail romeo
romeo has scheduled the find */ command
```

우에서 볼수 있는바와 같이 romeo는 관리자가 이 지령을 찾지 못하도록 FIND와 /사이에 많은 공백을 놓았다. 그러나 관리자는 이러한 경우까지도 고려하여 find와 /사이에 정규식 □*(*전에 공백이 있음)을 리용하였다. 사용자는 *가 공백문자에 대하여 없거나 혹은 하나이상의 공백문자를 대신한다는것을 생각해야 한다. 이 모든 스크립트들은 심중한 결함을 가지고 있다. 이것들은 패턴을 왜 찾지 못했는가 하는것을 지적하지 않는다. 지어는 찾으려는 파일들이 존재하지 않음에도 불구하고 not found 혹은 none of users통보문이 나타나며 2>에 의한 진단흐름에로의 방향절환은 grep와 awk의 통보문들이 말단에 나타나지 않게 한다. 리상적으로는 탐색에 앞서 파일이 있는가 하는것을 검사해야 한다. 여기서는 이러한 내용을 간단히 취급한다.

if문의 지령행에 놓인 조건은 이제부터는 조종지령이라고 언급하겠다. 임의의 unix지령은 if, while, until구조에 대한 조종지령일수 있다.

주해

모든 if문은 반드시 then문과 fi문을 가져야 하며 else문은 선택적이다. else문의 뒤로는 또 다른 if문(else if)을 포함시킬수 있으며 이러한 if문에 대한 매개 else문은 fi문으로서 반드시 닫겨져야 한다. elif문은 이와 같은 끌을 표현하는 문을 요구하지 않으며 정밀한 코드를 생성한다

18.9 if의 비교기능(test, [])

식을 평가하기 위하여 if문을 리용할 때 흔히 test명령이 그의 조종지령으로서 사용된다. test는 자기의 오른쪽에 있는 조건을 평가하기 위하여 연산자들을 사용하며 참 또는 거짓의 완료값을 돌린다. 이 완료값은 판단을 위한 if문에서 리용된다. test는 세가지 형태로 동작한다.

- 두수를 비교한다.
- 두개의 문자렬을 비교하거나 null값에 대하여 하나의 문자렬을 비교한다.
- 파일의 속성들을 검사한다.

이러한 검사는 쉘의 다른 명령문들과 함께 test문으로써 진행할수 있다. 쉘은 아무러한 출력자료도 표시하지 않으며 단지 파라메터 $?를 설정하는 값을 되돌린다. 다음의 항목들에서는 실제로 이러한 값을 검사한다.

18.9.1 수값비교

test에서 리용되는 비교연산자들(표 18-2)은 흔히 보던것과는 좀 다른 형식을 가진다. 이것들은 언제나 -(이음표)로 시작하며 그뒤에 두 문자단어가 오며 량옆에는 각각 공백이 놓인다. 아래에 전형적인 연

산자들을 보여 준다.

표 18-2. test에서 리용되는 수값연산자들

연산자	의미
-eq	같다
-nq	같지 않다
-ne	같지않기
-gt	크다
-ge	크거나 같다
-lt	작다
-le	작거나 같다

이 연산자들은 기억하기가 쉽다. 즉 -eq는 equal to를, -gt는 greater than을 의미한다. 사용자는 쉘에서의 수값비교가 옹근수값만으로 제한되며 소수점수값에 대해서는 소수부가 잘린다는것을 알아야 한다.

그러면 실례로 세개의 변수들에 어떤 값들을 할당하여 그것들이 서로 같은가 같지 않은가 하는 것을 검사하여 보자. 마지막 test문은 수값비교가 옹근수들에만 국한된다는것을 명백히 보여 준다.

```
$ x=5;y=7;z=7.2
$ test $x -eq $y ; echo $?
1
$ test $x -lt $y ; echo $?
0
$ test $z -gt $y ; echo $?
1
$ test $z ?eq $y ; echo $?
0
```

우에서는 test문을 단독으로 사용하였으며 그것은 if의 조종지령으로서 쓸수 있다.

다음의 스크립트는 test문과 특수한 쉘변수 $#를 리용한다. 스크립트는 단지 정확한 개수의 인수들이 입력되였는가를 검사한다.

```
$ cat arg_number_check.sh
if test $# -ne 3 ; then
     echo "You didn't enter three arguments"
else
echo "You entered the right number"
fi
```

이 스크립트를 두번 실행시키시오. 즉 한번은 인수 한개만으로서 다음은 세개의 인수로서 실행시키시오.

```
$ arg_number_check.sh 1024
You didn't enter three arguments
$ arg_number_check.sh /home list.tar 1024
You entered the right number
```

우에서 볼수 있는바와 같이 test를 리용하여 수값비교를 진행할수 있다. 그러면 awk로서 패턴을 찾

는 문제에 대하여 보자. test문을 다음과 같이 쓸수 있겠는가?

```
if test awk -F: '$1 ~ '/$1' { print }' /etc/passwd | wc -c -ne 0
```

여기서 test는 많은 인수들을 가지고 동작하며 오유를 내보낸다. wc역시 -ne를 항목으로서 해석하기때문에 오유를 발생시킨다. 우리가 검사해야 할것은 관흐름에 대한 출력자료(수값)이지 그의 돌림값은 아니다. 그러므로 wc의 문자계수값을 출력하기 위해서는 지령치환을 리용해야 하며 test로 그 값을 검사해야 할것이다. 아래의 지령행은 그러한 처리를 진행한다.

```
if test `awk -F: '$1 ~ '/$1/'{ print }' /etc/passwd | wc -c` -ne 0
```

여기서는 관흐름을 역인용부호(`)로 둘러 막았다. 우의 if문은 선택된 행을 화면에 표시하지 않는다. 다음의 스크립트에서 이러한 문제를 해결할것이다.

```
$ cat emp3b.sh
#!/bin/sh
if test $# -ne 1; then                # 1인수는 입력되지 않았다
    echo "Usage: $0 pattern"; exit 3
else
    if test `awk -F: '$1 ~ '/$1/'' /etc/passwd \
    | tee /dev/tty | wc -c` -ne 0                # 화면에도 현시
then
            echo "Pattern found - Job Over"
    else
            echo "Pattern not found"; exit 2
    fi
fi
```

여기서 두개의 if구조들을 볼수 있다. 하나의 if구조는 다른 if구조에 종속되여 있으며 매개는 fi문으로서 끝났다. 여기서 awk는 좀 다르게 사용되였다. 즉 awk는 선택조건만이 지적되는 경우 기정으로 자료를 출력하므로 여기서는 { print }문을 생략하였다. 또한 화면에 출력자료를 표시하기 위하여 지령 tee를 리용하였다. \은 두개의 행에 나타나는 세개의 지령에 대한 관흐름을 분할한다. wc -c는 awk의 출력자료(tee에 의하여 려과된)를 검사한다. 즉 계수값이 령이면 패턴이 없다는것을 의미한다. 우의 스크립트는 다음과 같이 동작한다.

```
$ emp3b.sh
Usage: emp3b.sh pattern
$ emp3b.sh image
image:x:502:100:The PPP server account:/home/image:/bin/ksh
Pattern found - Job Over
```

결론적으로 하나의 지령과 그리고 grep와 awk로서 탐색이 가능하도록 하는 지령치환과 함께 if를 리용할수 있다.

test의 생략

test문은 대단히 많이 사용되며 이것을 위한 생략방법이 존재한다. 즉 test문은 식을 둘러 싸는 한쌍의 꺾쇠괄호로 대신할수 있다. 다음의 두 형태는 동일하다.

```
test $x -eq $y
[ $x -eq $y ]
```

우에서 보는바와 같이 [ ]안쪽에 공백들이 반드시 있어야 한다. 두번째 형식은 조종하기가 더 쉬우며 앞으로는 이 형태를 리용한다. 여기서 공백(whitespace)에 대한 리용에 구애되지 않는다는것을 잊지 마시오.

주해

if x(여기서 x는 변수)와 같은 조건은 대부분의 프로그람작성언어들에서 리용된다. x가 0보다 크면 명령은 참으로 된다. 여기서도 같은 론리를 적용할수 있으며 if[$x -gt 0]의 생략형식으로서 [$x]를 쓸수 있다.

참고

수값비교에서는 $가 붙은 변수이름들을 인용부호로 묶지 않았다. 그러나 문자렬비교인 경우에는 좀 다르다. 만일 변수의 값이 한개의 단어라면 인용부호를 붙이지 않아도 되지만 여러 단어인 경우에는 인용부호를 반드시 붙여야 하며 그렇지 않으면 스크립트실행은 실패로 된다. 더우기 변수가 null문자렬을 가지고 있는 경우 인용부호를 붙이지 않으면 그러한 변수들에 대하여 오유가 발생할것이다. 사용자는 이러한 오유가 일어 날수 있는 곳들에 인용부호를 붙이는데 습관되여야 한다. 마찬가지로 변수들과 비교되는 문자렬들에 대해서도 "$ file"="j"와 같이 인용부호를 붙이시오.

18.9.2 문자렬비교

test는 또 다른 연산자들에 의하여 문자렬을 비교하는데 리용된다. 같다는것은 연산자 =로서 표현하며 같지 않다는것은 C언어형식의 연산자 !=으로서 표현한다. 다른 test연산자들과 같이 이 연산자들 역시 량옆에 공백을 놓아야 한다. 표 18-3은 문자렬처리검사에 대하여 보여 준다.

표 18-3. test에 의하여 리용되는 문자렬검사

검 사	아래와 같은 경우에 참이다
s1 = s2	문자렬 s1 = s2
s1 != s2	문자렬 s1이 s2와 같지 않다
stg	문자렬 stg는 할당되였으며 비지 않았다
-n stg	문자렬 stg는 빈 문자렬이 아니다
-z stg	문자렬 stg는 빈 문자렬이다
s1 == s2	문자렬 s1 = s2(Korn과 bash에서만)

다음의 스크립트는 C나 Java프로그람개발자들에게 쓸모 있는것이다. 사용되는 항목에 따라서 스크립트는 변수 file에 마지막으로 변경된 C 혹은 Java프로그람을 보존한다.

그다음 그 프로그람을 콤파일한다. 사용자는 이 스크립트에 한개의 인수 즉 파일의 형을 제공해야 한다. 인수는 c(C파일) 혹은 j(Java파일)일수 있다.

```
$ cat compile.sh
#!/bin/sh
```

```
if [ $# -eq 1 ] ; then
    if [ $1="j"] ' then
            file=`ls -t *.java | head -l`
            javac $file
    elif [ $1="c"] ; then
            file=`ls -t *.c | head -`
            cc $file && a.out
    else
            echo "Invalid file type"
    fi
else
    echo "Usage: $0 file_type\nVaild file types are c and j"
fi
```

javac와 cc는 각각 Java와 C프로그람들의 콤파일러들이다. 이 스크립트는 인수의 개수가 1인 경우에만 $1을 검사하는것으로써 처리를 계속한다. 만일 인수의 개수가 1이 아니면 이 스크립트에 대한 사용법을 표시하고 탈퇴한다. 우리들은 파일 a.out(c콤파일러에 의해 산생되는 기정적인 파일)를 실행하기 위하여 cc의 완료값을 사용해 본적이 있다. 그러면 이 스크립트를 한번 실행시켜 보자.

```
$ compile.sh
Usage: compile.sh file_type
Valid file types are c and j
$ compile.sh c
hello world
```

우에서 보는바와 같이 마지막으로 수정된 C프로그람이 실제로 printf문을 포함하고 있다는것을 알수 있다. 그러면 스크립트 그자체가 마지막으로 수정된 프로그람파일을 식별하여 사용자의 입력이 없이도 자동적으로 적당한 콤파일러를 선택할수 있다면 더 좋지 않겠는가? 이 문제에 대해서는 case명령문을 배운후에 론의하자.

그러면 입력값이 null값이라는것을 어떻게 검사하는가를 보기로 하자. 사용자가 실제로 문자렬을 입력하였는지 혹은 간단히 [Enter]건을 눌렀는지를 검사하기 위하여 다음의 스크립트에 문자렬비교기능을 리용하여 보자.

```
$ cat emp4.sh
#!/bin/sh
echo "Enter the string to be searched: \c"
read pname
if [ -z "$pname"] ; then                # -z는 빈 문자렬에 대하여 검사한다
    echo "You have not entered the string " ; exit 1
else
```

```
    echo "Enter the file to be used:\c"
    read flname
    if [ ! -n "flname"] ; then              # ! -n은 -z와 같다
            echo "You have not entered the filename" ; exit 2
    else
            grep "$pname" "$flname" || echo "Pattern not found"
    fi
fi
```

test의 결과는 !연산자에 의하여 반전된다. 스크립트의 실행은 두개의 위치에서 즉 패턴과 파일이름을 받아 들이는 위치에서 정지된다. null문자렬에 대한 검사는 두가지 방법으로써 진행할수 있다.

```
[ -z "$x"]
[ ! -n "$x"]
```

우의 두가지 경우는 같은 결과를 주는 서로 다른 방법이다. 스크립트는 입력자료중 어느 하나가 null문자렬이면 실행을 중지한다.

```
$ emp4.sh
Enter the string to be searched: director
Enter the file to be used: [enter]
You have not entered the filename
$ emp4.sh
Enter the string to be searched: director
Enter the file to be used: emp1.lst
1006|gordon lightfoot|director |sales     |09/03/38|140000
6521|derryk o'brien  |director |marketing |09/26/45|125000
```

또한 -a(AND)와 -o(OR)연산자들을 써서 같은 행에서 하나이상의 조건식을 검사할수 있다. 그러면 이 기능을 리용하여 앞의 스크립트를 간단화하자. 먼저 첫번째 if구조를 제거한 다음 두개의 문자렬을 받아 들인후에 아래와 같은 스크립트를 리용하자.

```
if [ -n "$pname" -a -n "$flname" ] ; then
    grep "$pname" "$flname" || echo "Pattern not found"
else
    echo "At least one input was a null string" ; exit 1
fi
```

test출력은 두 변수가 다 null이 아닌 문자렬일 때만 참이며 사용자는 스크립트가 두번 정지될 때 공백이 아닌 문자렬을 입력한다.

18.9.3 파일검사(test)

test는 여러가지 파일속성을 검사하는데 사용할수 있다. 실례로 파일이 읽기, 쓰기, 실행허가권을 가지고 있는가 하는것을 검사할수 있다. perl을 제외하고 다른 언어들에서는 표 18-4에서 보여 주는것과 같은 정교한 검사목록을 찾아 볼수 없다.

표 18-4. test에 의한 파일속성검사

검 사	파일이면 참
-f fname	fname은 존재하며 정규파일이다
-r fname	fname은 존재하며 읽기가능하다
-w fname	fname은 존재하며 쓰기가능하다
-x fname	fname은 존재하며 실행가능하다
-d fname	fname은 존재하며 등록부이다
-s fname	fname은 존재하며 비지 않았다
-e fname	fname은 존재한다(Korn과 bash에서만)
-u fname	fname은 존재하며 SUID비트설정을 가진다
-k fname	fname은 존재하며 점착비트설정을 가진다
-L fname	fname은 존재하며 기호련결이다(Korn과 bash에서만)
f1 -nt f2	f1은 f2보다 시간적으로 볼 때 더 최근에 작성되였다(Korn과 bash에서만)
f1 -ot f2	f1은 f2보다 더 오래전에 작성되였다(Korn과 bash에서만)
f1 -ef f2	f1은 f2와 련결된다(Korn과 bash에서만)

다른 언어들에서는 이 파일속성들을 쓸수 없든가 아니면 많은 량의 프로그람작성코드에 의하여 검사된다. test에서 쓰이는 파일검사명령문은 아주 치밀하다. 그러면 이 명령문을 리용하여 프롬프트에서 파일 emp.lst 의 몇가지 속성들을 검사해 보자.

```
$ ls -l emp.lst
-rw-rw-rw-    1 romeo    group       870 Jun  8 15:52 emp.lst
$ [ -f emp.lst ] ; echo $?
0                보통의 파일
$ [ -x emp.lst ] ; echo $?
1
$ [ ! -w emp.lst ] || echo "False that file is not writable"
False that file is not writable
```

여기서 !는 반전기호이며 따라서 [! -w file]은 [-w file]을 부정한다. 이 기능을 리용하여 인수로서 파일이름을 받아서 그것에 대하여 몇가지 검사를 진행하는 스크립트를 설계할수 있다.

```
$ cat filetest.sh
#!/bin/sh
if [ ! -f $1 ] ; then
    echo "File does not exist"
```

```
elif [ ! -r $1 ] ; then
    echo "File is not readable"
elif [ ! -w $1 ] ; then
    echo "File is not wirtable"
else
    echo "File is both readable and writable"
fi
```

그러면 두개의 파일 즉 존재하는 파일과 존재하지 않는 파일로서 스크립트를 검사해 보자.

```
$ filetest.sh emp3.lst
File does not exist
$ filetset.sh emp.lst
File is both readable and writable
```

일부 사람들은 test에 쉘의 통용기호를 리용하는 오유를 범한다. 실례로 주어 진 패턴과 정합되는 파일이 쓰기가능한가 하는것을 검사하기 위하여 [-w inde* .html]을 리용할수 없다.

18.10 case조건문

case문은 쉘에서 제공되는 두번째 조건명령문이다. perl을 포함하는 대부분의 언어들에서는 병렬조건명령문을 가지지 않는다. 이 명령문은 실제로 하나의 표현식을 여러개의 패턴들과 비교하며 다중분기를 위한 엄밀한 구조를 리용한다. 또한 통용기호를 가진 문자렬도 비교한다. 이 명령문의 문법적구조는 다음과 같다.

```
case 표현식 in
    패턴1)  지령묶음1 ;;
    패턴2)  지령묶음2 ;;
    패턴3)  지령묶음3 ;;
      ......
esac
```

case문은 먼저 표현식을 패턴1과 비교한다. 만일 비교가 성공하면 그에 따르는 지령을 실행시키며 이 지령은 하나 혹은 그이상일수도 있다. 비교가 실패하면 다음으로 패턴2가 비교된다. 매 지령목록은 한쌍의 반두점으로 끝나며 전체 구조는 esac으로 닫겨 진다.

그러면 파일체계의 몇가지 중요한 자료를 표시하는 간단한 스크립트를 사용하여 보자. 이것은 네개의 항목들을 가지는데 여러 행의 echo명령문에 의하여 표시된다.

```
$ cat filesys.sh
#!/bin/sh
tput clear
```

```
echo "\n 1. Find files modified in last 24 hours\n 2. The free disk space
  3. Space consumed by this user\n 4. Exit\n\n SELECTION: \c"

read choice
case $choice in
    1) find $HOME -mtime -l -print ;;
    2) df ;;
    3) du ?s $HOME ;;
    4) exit ;;
    *) echo "Invalid option"
esac
```

case는 $choice의 값을 문자렬 1, 2, 3, 4와 비교한다. 첫 세개의 항목들은 각각 find, df, du지령들을 실행한다. 항목 4는 프로그람을 탈퇴시키며 마지막항목(*)은 앞의 항목들과 비교되지 않는 임의의 항목들에 해당된다. 이 기능은 후에 효과적으로 리용될것이다.

파일들이 디스크에서 얼마나 많은 공간을 차지하는지 알려면 스크립트를 실행시키고 항목 3으로 선택하시오.

```
$ filesys.sh
    1. Find files modified in last 24 hours
    2. The free disk space
    3. Space consumed by this user
    4. Exit
SELECTION: 3
26944 /home/sumit
```

sumit는 자기의 홈등록부에서 우와 같은 많은 블로크(매개가 512byte까지)의 디스크공간을 리용하고 있다. 이와 같은 론리는 if문을 리용하여 실현할수도 있지만 case는 명백히 말해서 더 정밀하다. 그러면 이제부터 case문의 다른 특징들을 보기로 하자.

18.10.1 다중패턴정합

case는 egrep형식으로 둘이상의 패턴을 비교할수 있다. 만일 사용자가 수요일과 금요일에는 파일 혹은 등록부에 대한 완전한 여벌복사를 그리고 다른 날들에는 증가적인 여벌복사를 진행하는 여벌체계구조를 가지고 있다면 case는 이 론리를 실현하기 위한 아주 엄밀한 구조를 제공한다. 날자는 date지령의 출력자료로부터 자를수 있기때문에 다음의 스크립트에서는 case문에 문자렬을 제공하기 위하여 지령대입을 리용한다.

```
$ cat back.sh
case `date | cut -d " " -f1` in                # 3개의 문자로 된 날을 출력한다
```

```
    Web|Fri) tar -cvf /dev/fd0 * ;;
          *) find . -newer .last_full_backup_time -print > tarilist
             tar -I tarilist -cvf /dev/fd0 ;;           #Solaris에서의 -I선택항목
esac
```

date지령출력의 첫번째 마당은 날자를 표시하며 여기서는 case에 입력자료를 제공하기 위하여 cut지령으로 그것을 선택하였다. 첫번째 항목은 egrep와 awk와 같은 두개의 패턴을 보여 준다. case문은 여러개의 패턴을 비교하는 경우 패턴구분문자로서 |을 리용한다. 이 항목은 현재의 요일을 문자렬 wed 혹은 fri와 정합시킨 다음 tar지령을 실행한다. 여기서 우리들은 다른 요일들을 정합시키기 위하여 고심할 필요는 없다. *는 정합되지 않는 모든 항목들을 정합시킨다.

스크립트는 임의의 날에 기동시킬수 있으며 지령은 자동적으로 실행될것이다. tar지령이 완전한 여벌복사와 증가적인 여벌복사를 어떻게 수행하는가 하는데 대해서는 22.10.2에서 구체적으로 론의한다.

그러면 또 다른 실례를 보자. 프로그람작성자들은 흔히 y와 Y 혹은 n과 N에 대하여 사용자응답을 검사해야 할 필요성이 제기된다. if로 이 론리를 실현하려면 복합조건기능을 리용해야 할것이다.

```
if [ "$choice"= "y" -o "$choice"= "Y" ]
```

case문을 리용한다면 대단히 간단해 진다. 즉 case문에 표현식 y|Y를 리용하면 대소문자를 다 정합할수 있다.

```
echo "Do you wish to continue? (y/n): \c"
read answer
case "$answer" in
    y|Y) ;;             NULL명령문 즉 수행되는 동작은 없다
    n|N) exit ;;
      *) echo "Invalid option" ;;
esac
```

18.10.2 case의 통용기호리용

case는 통용기호를 사용하는 대단히 우월한 문자렬정합기능을 가지고 있다. 즉 메타문자 *, ?와 문자모임(8.2)을 정합시키는 파일이름을 리용한다. 그러나 이것은 현재등록부의 파일 그자체가 아니라 오직 문자렬들을 정합시킨다. 앞의 실례에 대한 수정된 case구조는 사용자가 여러가지 방법으로 질문에 대답하게 한다.

```
case "$answer" in
     [yY][eE]*) ;;            YES, yes, Yes 등을 정합한다
      [nN][oO]) exit ;;             NO, no, nO, No를 정합한다
             *) echo "Invalid response"           기타 경우에 대해서는 무시한다
esac
```

첫 두개의 항목들에서 통용기호사용은 단순하면서도 충분한 내용을 담고 있다. *는 두개의 항목들에서 나타나며 매 항목들에서 다른 의미를 가진다는것을 생각하시오. 첫 항목에서 *는 보통의 통용기호로서 동

작하며 마지막항목에서 정합되지 않은 모든 항목들에 대해서는 거절한다. 마지막 case항목에는 ;;이 필요없지만 원한다면 리용해도 일 없다. ?를 리용하면 ????으로서 네 문자로 이루어 진 문자렬을 정합할수 있다. 그러나 반드시 수자들만을 포함하여야 한다면 [0-9][0-9][0-9][0-9]가 정확할것이다. 다음의것은 말단으로부터 6문자수자마당을 어떻게 유효하게 하는가 하는것을 보여 준다.

```
n="[0-9][0-9][0-9][0-9][0-9][0-9]"
echo "Enter a date string\c"
read dstring
case $dstring in
   ???????*) echo "String exceeds six characters" ;;
         $n) echo "A six character numeric field" ;;
          *) echo "The string is either non-numeric"
             echo "or less than six characters long" ;;
esac
```

지금까지 강력한 문자렬정합기능을 보았는데 그러면 compile.sh스크립트(18.9.2)를 좀 더 강력하게 만들수는 없겠는가? 수정될 프로그람은 마지막으로 변경된 C 혹은 Java프로그람을 콤파일해야 하며 자동적으로 콤파일러를 선택하여야 한다.

```
$ cat compile2.sh
file=`ls -t *.java *.c 2>/dev/null | head -l`
case $file in
       *.c) cc $file && a.out ;;
    *.java) javac $file ;;
         *) echo "There's no java or C program in the current directory"
esac
```

첫번째 명령문은 대부분의 작업을 포함하고 있다. 이것은 마지막으로 변경된 C 혹은 Java파일을 선택하여 변수 file에 그 값을 보존한다. 다음으로 case문에 대한 처리를 진행한다. 그리고 확장자를 비교하여 정확한 콤파일러를 호출한다. 이것은 모두 6개 행의 코드로 되여 있다.

주해

*는 일반적으로 통용기호로서 동작한다. 그러나 case문에서는 두가지 방식으로 사용된다. 패턴에 매몰된 별표(*)는 임의의 개수의 문자들을 정합하지만 * 단독으로는 이전 항목들에서 정합되지 않은 임의의것에 대하여 마지막 case항목으로서 놓여 진다.

18.11 계산과 문자렬처리(expr)

Bourne쉘은 옹근수값의 크기비교는 진행할수 있지만 그 어떤 계산기능도 가지고 있지 않다. 이러한 원인으로 하여 Bourne쉘은 계산기능에 대하여 expr외부지령에 의존한다. 이 지령은 두가지 기능들을 하나로 결합하고 있다.

- 옹근수들에 대한 산수연산을 수행한다.
- 문자렬들을 관리한다.

여기서는 이 두가지 기능들을 실현하기 위하여 expr를 리용하지만 그것은 문자렬처리를 진행할 때 대단히 읽기 불편한 코드를 제공한다. Korn쉘이나 bash쉘을 리용하면 이것들을 처리하기 위한 더 좋은 방법(19.8)을 가질수 있다.

18.11.1 연산기능들

expr지령은 네가지 기본산수연산은 물론 나머지연산도 수행할수 있다.

```
$ x=3 ; y=5
$ expr 3 + 5
8
$ expr $x - $y
-2
$ expr 3 \* 5
15
$ expr $y / $x
1
$ expr 13 % 5
3
```

여기서 연산자(+, -, *,...)의 량옆에는 반드시 공백이 있어야 한다. 곱하기연산자만은 쉘이 그것을 파일이름메타문자로 해석하지 않도록 의미해제시켜야 한다. expr지령이 옹근수만을 조종하므로 나누기의 경우에는 옹근수부만을 준다.

expr지령은 변수를 할당하기 위한 지령치환으로서 흔히 사용된다. 실례로 두 수값의 합으로 변수 z를 설정할수 있다.

```
$ x=6 ; y=2; z=`expr $x + $y`
$ echo $z
8
```

expr는 일반적으로 변수값을 증가시키는데 리용된다. 모든 프로그람언어들은 변수의 값을 증가시키는 생략방법을 가지고 있으며 UNIX 역시 이러한 방법을 가져야 하는것은 당연한 일이다.

```
$ x=5
$ x=`expr $x + 1`                C의 x++와 같다
$ echo $x
6
```

만일 Bourne쉘을 리용한다면 자기의 쉘스크립트들에 expr지령의 이러한 사용법을 리용해야 할것이다.

18.11.2 문자렬처리

expr의 문자렬처리기능이 명백히 째여 있지는 못해도 Bourne쉘사용자들은 대체로 이것을 리용한다.

문자렬을 관리하기 위하여 expr는 두점으로 구분되는 두개의 표현식을 리용한다. 작업의 대상으로 되는 문자렬은 :의 왼쪽에 놓이며 정규식은 그의 오른쪽에 놓인다. 정규식의 본질로부터 expr는 세개의 중요한 문자렬처리을 수행할수 있다.

- 문자렬의 길이를 결정
- 부분문자렬을 추출
- 문자렬에서 문자의 위치를 탐색

문자렬의 길이

문자렬의 길이는 아주 간단히 얻을수 있다. 정규식 .*는 패턴과 정합되는 문자들의 개수 즉 전체 문자렬의 길이를 표시해야 한다는것을 지령 expr에 알린다.

```
$ expr "robert_kahn" : '.*'                :주위에 공백이 있음
11
```

여기서 expr는 임의의 문자(.*)의 출현회수를 계수하였으며 그 결과는 grep나 sed에서 리용하였을 때의 결과와는 완전히 다르다. 이 기능은 자료의 유효성검사에 대단히 쓸모 있다. 건반을 통하여 받아 들인 이름이 20문자를 넘지 않는 사람들에 대해서만 유효화하자. 아래의 expr지령렬은 이와 같은 과제에 대단히 능숙하다.

```
echo "Enter your name: \c"
read name
if [ `expr "$name" : '.*'` -gt 20 ] ; then
    echo "Name too long
fi
```

부분문자렬의 추출

expr는 기호렬 \(와 \)에 의하며 닫겨 진 문자렬을 선택할수 있다. 만일 4자리의 수값문자렬에서 2자리의 수값문자렬만을 선택하려 한다면 패턴묶음을 만들어야 하며 다음의 방법으로 그것을 추출할수 있다.

```
$ stg=2001
$ expr "$stg" : '..\(..\)'                마지막 두개의 문자를 추출한다
01
```

여기서 패턴묶음은 \(..\)이다. 이것은 sed(15.12)에서 사용하였던 꼬리표 붙은 정규식(TRE)이지만 여기서는 좀 다른 의미를 가진다. 이것은 $stg의 값에서 첫 두 문자는 무시되여야 하며 세번째 문자위치에서부터 선택되여야 한다는것을 의미한다(여기서 \1이나 \2는 리용되지 않는다).

문자의 위치탐색

expr는 문자렬내에서 지적된 문자의 첫 발생위치를 돌릴수 있다. $stg라는 문자렬값에서 문자 b의 위치를 찾자면 그앞에 b가 아닌 문자([^ b]*)들의 개수를 계수해야 한다.

```
$ stg="paul_baran"
$ expr "$stg" : '[^b]*b
6
```

또한 test명령문의 일부 기능들이 expr지령에서 중복되며 같은 방법으로 수값비교연산자들을 사용한다. test가 쉘의 내부기능이고 속도가 대단히 빠르므로 여기서는 취급하지 않는다. Korn쉘과 bash는 내부적인 연산기능과 문자렬처리기능을 가지고 있다. 즉 이러한 쉘들에서는 expr지령이 필요 없으며 다음장에서 취급된다.

18.12 각이한 이름에 의한 스크립트의 호출($0)

7.13과 련관시켜 본다면 우리는 여러가지 이름으로 파일을 호출하고 또 호출되는 이름에 따라서 서로 다른 처리를 진행하는 현실적인 가능성을 가지게 되였다. 명백히 말해서 이러한 처리를 진행하는 몇개의 UNIX지령들이 있다. 우리는 case를 어떻게 사용하며 expr로서 문자렬을 어떻게 처리하는가 하는것을 알고 있으며 마지막 Java프로그람을 콤파일, 편집, 실행시키기 위한 단일한 스크립트를 설계한적이 있다. 스크립트파일은 세개의 이름을 가질것이며 그것을 개발하기전에 프로그람작성의 몇가지 본질적인 측면들을 리해하자.

Java프로그람은 .java확장자를 가져야 한다. javac파일이름형식으로 Java프로그람이 콤파일될 때 .class파일이 생긴다. 그러나 java지령으로 프로그람을 실행시킬 때는 확장자를 리용하지 말아야 한다. 즉 파일 hello.java를 콤파일하여 hello.class를 만들고 그다음 java hello로써 프로그람을 실행시켜야 한다. 따라서 확장자를 없애고 기본파일이름만을 선택할수 있어야 하며 expr를 리용하면 이러한 처리는 대단히 간단하다.

우리는 장의 첫 부분에서 파라메터 $0을 평가할 때 그앞에 ./을 붙이는 bash에 대하여 취급하였다. 그러면 쉘로서 bash를 리용하는 경우 ./을 제거하여 쉘에 독립인 스크립트를 작성하여 보자. 이 모든 기능을 수행하는 스크립트 comj을 아래에 보여 준다.

```
$ cat comj
# Script that is called by different names
lastfile=`ls -t *.java | head -l`


case $SHELL in
/ksh|/sh) command=$0 ;;
    bash) command=`expr $0 : `./\([^/]*\)'` ;;                # ./을 제거한다
esac
case $command in
     runj) lastfile=`expr $lastfile :'\(.*\).java'`              # .java를 제거한다
            java $lastfile ;;
      vij) vi $lastfile ;;
     comj) javac $lastfile && echo "$lastfile comiled successfully" ;;
esac
```

이 스크립트는 두개의 case명령문을 가지고 있다. 첫번째 case문에서는 $0으로부터 스크립트의 이름을 취한다. Bourne과 Korn쉘들에 대해서는 변수 command에 $0을 대입하는것외에는 아무것도 하는것

이 없다. bash에 대해서는 expr지령에 의하여 ./이 제거된다. expr는 기호 /이 따르는 임의의 개수의 문자렬을 의미하는 묶음 .*/을 무시한다. 다음 비사선문자묶음을 선택한다. 이것은 변수 command에 기억된 지령이름만을 효과적으로 뽑아 낸다.

두번째 case문은 프로그람이 호출되는 이름을 검사한다. 첫 항목(runj)은 .java확장자를 제거한 기본이름만을 추출한다. 마지막으로 변경된 파일이름이 변수 lastfile에 보존되여 있으므로 나머지처리는 대단히 간단해 진다. 마지막으로 할것은 ln지령으로 comj에 대한 련결을 만드는것이다.

```
ln comj runj
ln comj vij
```

이렇게 하면 프로그람의 편집, 콤파일, 실행을 위하여 각각 vij, comj, runj를 실행시킬수 있다. 여기서는 프로그람을 콤파일하기만 하자.

```
$ comj
hello.java compiled successfully
```

얼마나 간단한가? vi사용시에 이 과제들에 두개의 기능건을 할당하여 더 간단하게 만들수 있다. 파일 .exrc에 아래와 같은 두 행을 놓으시오.

```
:map #1 :w^M:!comj^M              w^M으로 먼저 파일을 보관한다
:map #2 :!runj^M
```

프로그람을 편집하는 도중에 F1을 눌러서 프로그람을 콤파일할수 있으며 F2건을 눌러서 실행시킬수 있다. UNIX가 아닌 다른데서 이러한 조작을 할수 있겠는가?

18.13 sleep와 wait

쉘스크립트에서 처리에 앞서 사용자가 화면에서 어떤 통보문을 볼수 있도록 일정한 시간지연을 줄 필요가 있다. 또한 사건발생(실례로 파일이 존재상태로 되는것과 같은것)에 대하여 규칙적인 검사(실례로 1분에 한번)를 진행할 필요도 있다. sleep는 이러한 시간적지연을 주기 위한 UNIX지령이다. 이 지령은 지연시간을 가리키는 인수와 함께 리용된다. 즉 인수로서 몇초동안 실행이 중지되는가 혹은 잠자기(sleeping)되는가 하는 지연시간을 지적한다.

```
$ sleep 100 ; echo "100 seconds have elapsed"
100 seconds have elapsed
```

통보문은 지령이 호출된 때로부터 100초동안 화면에 표시된다. 이 지령의 특수한 기능은 잠자기되는 동안은 overhead에 빠지지 않는다는것이다.

wait는 모든 배경일감들이 완성되였는가를 검사하는 쉘내부지령이다. 이 지령은 배경에서 일감을 실행시키고 그것이 완성되여 또 다른 지령을 실행할수 있는가 하는것을 확인하려는 경우 대단히 쓸모 있다. 즉 마지막배경일감이 완성될 때까지 대기하도록 wait지령을 리용할수 있다.

18.14 while순환과 until순환

지금까지 개발된 패턴을 입수하는 스크립트들중에서 그 어느것도 사용자에게 실패응답을 수정할 다시한번의 기회를 주지 않는다. 순환은 명령묶음을 반복적으로 수행하게 한다. 쉘은 세가지 형태의 순환 즉 while, until, for를 가지고 있다. 이 모든것들은 그의 조종지령이 허락하는 회수만큼 열쇠단어들로 둘러 막힌 명령묶음을 반복한다.

while명령문에 대해서는 대부분의 프로그람작성자들이 알고 있을것이다. 이 명령은 그의 조종지령이 참인 완료값을 돌릴 때까지 명령묶음을 반복실행시킨다. 이 지령의 일반형식은 다음과 같다.

```
while 조건이 참인동안
do                열쇠단어
    지령묶음                  순환본체
done                열쇠단어
```

열쇠단어 do와 done으로 둘러 막힌 지령들은 조건이 참인 동안 반복적으로 실행된다. 조종지령으로서는 앞에서 취급된 임의의 UNIX지령이나 test문을 리용할수 있다.

그러면 흔히 쓰는 while순환응용프로그람에 의하여 리해해 보자. emp5.sh스크립트는 같은 행에서 하나의 코드와 서술문을 받으며 그 행을 파일 newfile에 쓰기한다. 다음 몇개의 입력을 요구하는 프롬프트를 내보낸다.

```
$ cat emp5.sh
#!/bin/sh
#Program: emp5.sh  0Shows use of the while loop
answer=y                 #순환의 시작으로서 y를 설정한다
while ["$answer"="y"]                 # 조종지령
do
    echo "Enter the code and description: \c"
    read code description                 #둘다 읽는다
    echo "$code|$description" >> newlist                 #행을 파일에 추가한다
    echo "Enter any more (y/n)? \c"
    read anymore
    case $anymore in
          y*|Y*) answer=y ;;                 # y 혹은 y로 시작하는 어떤것
          n*|N*) answer=n ;;                 # n 혹은 N으로 시작하는 어떤것
              *) answer=y ;;                 # 다른 응답은 y를 의미한다
     esac
done
```

우의 프로그람에서 설명문은 해석할 필요가 없다. 먼저 이것을 실행해 보자.

```
$ emp5.sh
Enter the code and description: 03 analgesics
Enter any more (y/n)? y
Enter the code and description: 04 antibiotics
Enter any more (y/n?)[Enter]              응답이 없다. 이 경우 y로 가정한다
Enter the code and description: 05 OTC drugs            세개의 단어가 있다
Enter any more (y/n)? n
```

그 어떤 특별한 조작이 없는 간단한 스크립트이다. 우에서 입력한 자료는 파일 newlist에 기록된다.

```
$ cat newlist
03|analgesics
04|antibiotics
05|OTC drugs
```

18.14.1 while을 리용하여 파일을 기다리기

흥미 있는 while순환응용프로그람을 보자. 프로그람 b가 다른 프로그람 a에 의하여 만들어 진 파일을 읽어야 할 경우가 있다. 프로그람 b는 a에 의하여 파일이 만들어 진 후에만 실행될수 있다. 다음실례에서 이와 같은 론리를 실현해 보자.

스크립트 monitfile.sh는 주기적으로 파일(여기서는 invoice. lst)의 존재성여부에 대하여 디스크를 감시하며 일단 파일을 찾기만 하면 alloc.p1스크립트를 실행한다.

```
$ cat monitfile.sh
while [ ! -r invoice.lst ]              # 파일 invoice.lst을 읽을수 없는동안
do
      sleep 60                  # 60초마다 파일을 탐색한다
done
alloc.pl                # while순환에서 탈퇴한후에 이 프로그람이 실행된다
```

파일 invoice가 읽혀 지지 않는한(! -r는 읽기불가능을 의미한다.) 순환은 계속 반복된다. 파일이 읽기가능으로 되면 순환은 끝나고 프로그람 alloc.p1이 실행된다. 이 스크립트는 아래와 같이 배경에서 실행되여야 하는 전형적인 프로그람이다.

```
monifile.sh &
```

여기서 파일의 존재성여부에 대하여 60초를 간격으로 검사하도록 sleep지령을 리용하였다.

18.14.2 사용자의 용량소비를 알아보기

22.10을 참고하여 지령 du -s /home/*의 출력을 보시오. 이 지령이 인수로서 /home/* 를 가지고 실행되면 매 사용자에 대한 디스크리용정보를 현시한다. 아래에 그에 대한 몇가지 실례를 보여 준다.

```
    166     /home/enquiry
```

```
4054    /home/henry
 647    /home/image
64308   /home/sumit
```

여기서는 홈등록부들이 등록부 /home에 보관된다고 생각한다. 이 출력자료의 매행을 읽기 위하여 while순환을 리용할수 있으며 지적된 수값 혹은 기정적으로 4000개의 블로크를 초과하는 모든 사용자에 대하여 뿌리에 통보문을 우편으로 보낸다. 이것은 스크립트 du.sh가 인수를 받지 않거나 오직 한개 받는다는것을 의미한다.

```
$ cat du.sh
# du.sh -- Program to monitor free space on disk
case $# in
    0) size=4000 ;;                #사용자입력이 없는 기정크기
    1) size=$1 ;;                 #호출에 의한 지정
    *) echo "Usage: $0 [blocks]" ; exit ;;
esac
du -s /hom/* | while read Blocks user
do
   [ $blocks -gt $size ] && echo "$user has consumed $blocks blocks"\
                          | mail root                #목록이 뿌리로 간다
done
```

여기서 while순환은 du지령의 표준출력을 입력으로 가지며 매행은 두개의 변수 blocks와 user로 읽혀 진다. 그다음 변수 block의 값과 수값 4000(인수가 지적되지 않으면) 혹은 지적된 인수의 값과 비교한다. 매행에 대하여 분할된 통보문은 뿌리에 우편으로서 보내진다. 이 지령을 두가지 방법으로 실행시킬수 있다.

```
du.sh
du.sh 8000                 8000개의 블로크가 넘는 사용자들만 선택한다
```

뿌리사용자가 자기의 우편함을 열면 한계를 초과한 모든 사용자에 대한 통보문이 제시된다. 이 통보문은 다음과 같다.

```
/home/sumit has consumed 64308 blocks
```

이 스크립트는 항시적으로 디스크공간을 검사해야 하며 또 그들이 리용하기로 되여 있는 디스크공간보다 더 많은 공간을 소비하는 사용자들을 식별해야 하는 체계관리자에게 대단히 쓸모 있다. 그리고 오전 10시부터 오후 7시사이에 3시간간격으로 그날 작업에 대하여 실행되여야 하는 crontab일감에 보다 효과적이다.

```
0 10,13,16,19 * * 1-5 /home/admin/scripts/du.sh              crontab입력
```

이 스크립트에는 대단히 큰 적재를 진행할 때 위험을 줄수 있는 약점이 있다. 이것이 동작할 때에는 매 사용자에 대한 개별적인 통보문을 보낸다. 사실상 이것은 전체 목록을 포함하는 단일한 우편통보문이여야 할

것이다. 즉 이것이 직관적이지 못하다 하더라도 열쇠단어 done에서 echo출력을 관련결했어야 할것이다.

```
done | mail root
```

우리는 다음장에서 방향절환에 대하여 두번째로 보게 된다. 때가 되면 이 스크립트는 우리의 목적에 맞게 수정될것이다.

18.14.3 무한순환의 설정

체계관리자로서 5분에 한번씩 자기의 디스크에서 리용가능한 빈 공간을 검사하려 한다고 하자. 이때는 무한순환을 리용해야 하며 가장 쉬운 방법은 while의 조종지령으로서 가상지령을 쓰는것이다. 이 지령은 true값외에는 아무것도 돌려 주지 않는다. 사실상 /bin에는 true라는 지령이 있으며 배경에 순환을 설정할수도 있다.

```
while true ; do
    df -t
    sleep 300
done &
```

일단 이 프로그람이 배경에서 기동되면 5분마다 df지령의 출력자료가 화면에 표시되는것외에는 작업에 아무러한 영향도 주지 않는다(6.17). 이러한 처리를 강제로 완료시키기 위해서는 지령 kill $!를 리용해야 한다. 이 지령은 마지막배경일감을 강제로 완료시킨다(10.10.2).

true지령과 비슷하게 항상 false값을 돌려 주는 지령 false도 있다. 이것들은 UNIX체계들에서 쓸수 있는 가장 단순한 지령들이다.

```
$ cat /bin/true
$ _                 파일에는 아무것도 없다. 돌림값은 true이다
$ cat /bin/false
exit 255
```

이 두 지령의 Linux판본들은 몇가지 기능을 더 가지고 있지만 여기서는 그것을 무시한다. 이 무한순환을 배경에서 실행시키지 않는다면 새치기건을 눌러서 그것을 완료시켜야 한다는것을 잊지 마시오.

순환안쪽에서는 break열쇠단어를 리용하여 탈퇴할수 있다. 두개의 열쇠단어 break와 coutinue는 순환에 리용되며 다음절에서 취급된다.

18.14.4 until순환

until명령문은 조건이 거짓을 유지하는 동안 순환본체가 반복실행되는 while구조에 대한 보충적인 명령문이다. 이것은 전체적인것을 볼수 있는 또 다른 간단한 방법이다. 어느 형식이나 표현식이 한 형식에서 다른 형식으로 절환되는 경우 그것을 반전시켜야 한다는 점을 제외하고는 서로 교차적으로 사용될수 있다. 어떤 사람들은 다음의 방법으로 이전의 while조종지령을 쓰는것을 더 좋아 한다.

```
until [ -r invoice.lst ]                invoice.lst가 읽기가능할 때까지
```

이 행은 "파일 invoice.lst가 읽기가능으로 될 때까지 …"으로 해석된다. 이 형식은 더 리해하기 쉽다.

18.15 실례스크립트

두가지 흥미 있는 스크립트를 개발하는것으로써 지금까지 배운 지식을 공고히 다져 보자. 하나는 파일이 목적지에 존재하면 수값확장자를 가진 파일을 제공하는 확장된 cp지령이다. 다른 하나는 유효성을 확인하는 자료입력스크립트이다.

이 스크립트들을 취급하기전에 쉘순환들에서 리용되는 두개의 열쇠단어 break와 continue를 알아야 한다. 이것들은 C언어와 Java언어들에서도 리용된다. continue명령문은 그뒤의 모든 명령들의 실행을 일시 정지시키고 다음반복을 위하여 순환의 제일 꼭대기로 조종을 바꾼다. break명령문은 조종이 순환밖으로 벗어 나게 한다. 이 명령문들도 인수를 가지고 동작한다.

18.15.1 수자확장자를 가진 파일로 바꾸기(cpback.sh)

중요한 파일을 어떤것으로서 덧쓰기하여 잃어 버린적이 있는가? 그림 18-1에 보여 주는 스크립트 cpback.sh는 cp지령에 의한 우연적인 덧쓰기로부터 파일을 보호한다.

```
# Program cpback.sh -- Copies a file to a directory
# Makes a backup instead of overwriting the destination file
# Copies foo to foo1.1 if foo exists or foo.2 if foo.1 exists .........
if [ $# -ne 2 ] ; then                              #이 스크립트에는 두 인수가 요구된다
    echo "Usage: $0 source destination"; exit
elif [ ! -d $2 ] ; then
    echo "Directory $2 doesn't exist"
else
    file=$1
    if [ ! -f $2/$file ] ; then                     #파일이 목적지에 없다
        cp $1 $2
    else
        copies=1                                    #검사시작
        while true ; do                             #수값확장자가 존재하는 동안
            if [ ! -f $2/$file.$copies ] ; then
                cp $1 $2/$1.$copies                 #수값확장자를 주고 복사한다
                echo "File $1 copied to $1.$copies"
                break                               #일감이 수행되였다. 순환을 완료한다
            else                                    #파일이 수값확장자를 가지면
                copies=`expr $copies + 1`           #또 존재하면 다음단계에로
            fi
        done
    fi
fi
```

그림 18-1. 덧쓰기없이 파일을 복사하기 위한 스크립트 cpback.sh

이 프로그람의 첫 판본은 두가지 인수로서 복사될 파일이름과 등록부이름을 받는다. 이것은 파일 foo가 목적지에 존재하는가를 검사하며 만일 존재한다면 foo.1에서 시작하는 수자확장자를 뒤에 붙인다.

이것 역시 존재할수도 있으므로 재쓰기없이 파일을 복사하기 위하여 수값을 반복적으로 증가시킨다.

break명령문은 파일이 복사되기만 하면 순환을 끝낸다. 그러면 등록부 safe를 만들고 이 등록부에 파일vvi.sh을 복사하여 보자.

```
$ cpback.sh
Usage: cpback.sh source destination
$ cpback.sh vvi.sh safe1
Directory safe1 doesn't exist
$ cpback.sh vvi.sh safe
$_                파일은 vvi.sh로서 복사되였다
```

아마 자기가 바라던대로 파일이 복사되였을것이다. 그러면 이러한 복사조작을 두번 반복하여 보자.

```
$ cpback.sh vvi.sh safe
File vvi.sh copied to vvi.sh.1
$ cpback.sh vvi.sh safe
File vvi.sh copied to vvi.sh.2
```

스크립트는 정확히 동작한다. 사용자는 이 스크립트를 리용하여 파일을 복사하여 개별적인 등록부에 프로그람의 모든 판본들을 다 보관할수 있다. 후에 이 스크립트는 여러개의 파일을 처리할수 있도록 변경될것이다.

18.15.2 자료입력스크립트(dentry1.sh)

그림 18-2에 보여 주는 다음의 스크립트 dentry1.sh는 말단으로부터 코드와 그에 대응하는 서술문을 받아서 몇가지 초보적인 유효성검사를 진행하고 그다음 파일 desig.lst에 입력자료를 추가한다. 이 스크립트는 입력된 코드가 파일에 이미 존재하는가 하는것을 검사하며 사용자가 정확한 응답을 할 때까지 반복하여 질문한다.

스크립트는 두개의 마당 즉 목적지코드와 서술문을 입력할것을 요구한다. 또한 두개의 순환고리를 리용하며 여기서 하나는 다른 순환고리에 포함되여 있다. 입력된 코드가 파일에 이미 존재하거나 두개의 수자로 구성되여 있지 않으면 다시 입력해야 한다. 마찬가지로 서술문이 공백을 제외한 비자모문자를 포함하고 있다면 다시 입력해야 한다(*[!\□a-zA-Z]*). continue명령문은 자료를 재입력시키거나 재생주기(fresh cycle)를 시작하게 한다. 내부순환에서 break명령문은 행을 추가한후에 순환에서 탈퇴한다.

스크립트와 대화를 해보면 이러한 론리가 납득이 될것이다.

```
$ dentry1.sh
Designation code: 01
Code exists
Designation code: 07
Description     : security officer


Wish to continue? (y/n): Y
```

```
#!/bin/sh
while echo "Designation code: \c" ; do                    #Linux의 -e를 리용한다
    read desig
    case "$desig" in
     [0-9][0-9]) if grep "^$desig" desig.lst>/dev/null ; then      #코드가 존재하면
                   echo "Code exists" ; continue                   #순환시작에로
              fi ;;
            *) echo "Invalide code"; continue ;;           #두자리 코드가 아니면
    esac

   while echo "Description  : \c" ; do
      read desc
      case "$desc" in
      [!\ a-zA-Z]*) echo "Can contain only alphabets and spaces"; continue ;;
              " ") echo "Description not entered" ; continue ;;
               *) echo "$desig|$desc" >> newlist ; break            #순환완료
    esac
   done

   echo "\nWish to continue? (y/n): \c"
   read answer
   case "$answer" in
     [yY]*) continue ;;                                              #순환시작에로
        *) break ;;                                                  #순환완료
    esac
done
```

그림 18-2. dentry1.sh스크립트

```
Designation code: 8
Invalid code                두자리 수여야 한다
Designation code: 08
Description     : vice president 1
Can contain only alphabets and spaces
Description     : vice president

Wish to continue? (y/n); n
```

파일 newlist의 내용을 보면 두개의 추가된 항목을 볼수 있다.

```
$ cat newlist
07|security officer
08|vice president
```

우리는 파일에 대한 검사, 수를 반복적으로 증가시키기, 사용자에게 유효입력자료에 대해 계속 질문하는것 등 이미 여러가지 방법으로 while순환을 리용하였다. 이것은 프로그람작성자모두가 정통해야 할 아주 중요한 구조이다.

18.16 목록에 의한 순환(for)

for순환은 다른 프로그람언어들에서 사용되는것과 구조상차이가 있다. BASIC사용자들에게는 새로운감(next문이 없고 간격값이 없는)을 주게 된다. while이나 until과는 달리 for문은 조건을 검사하지는 않지만 대신 목록을 리용한다. 이 구조의 문법적형식은 아주 간단하다.

```
for 변수 in 목록 ; do
      지령들                순환본체
done
```

순환본체는 열쇠단어 do와 done으로 둘러 막힌것으로서 지금까지 본 순환명령문들에서와 같다. 그러나 여기에는 추가적인 파라메터로서 변수와 목록이 있다. 순환본체는 목록에 항목들이 있다면 반복적으로 실행된다. 다음의 간단한 실례는 이것을 리해하는데 도움이 될것이다

```
$ for file in chap20 chap21 chap22 chap23 ; do
>   cp $file ${file}.bak
>   echo $file copied to $file.bak
>done
chap20 copied to chap20.bak
chap21 copied to chap21.bak
chap22 copied to chap22.bak
chap23 copied to chap23.bak
```

여기서 목록은 공백으로 구분되는 여러개의 문자렬로 이루어 진다. 목록에서 매 항목은 변수 file에 할당된다. 변수 file에는 먼저 chap20이 할당되고 다음 chap21, chap22, …의 차례로 할당된다. 매 파일은 .bak확장자를 가지고 복사되며 파일이 복사된후에 완성통보문이 현시된다.

지령행에서 여러개의 변수를 사용할수도 있다. 그것들은 순환을 진행하기전에 쉘에 의하여 값이 평가된다.

```
$ for var in $PATH $HOME $MAIL ; do echo "$var" ; done
/bin:/usr/bin:/home/local/bin:/usr/bin/X11:.:/oracle/bin
/home/romeo
/var/mail/romeo
```

한개 행에 전체 순환을 기입하고 싶은 경우에는 우와 같이 반드시 반두점으로 구분해 주어야 한다. 우의 매행들은 각각 세개의 환경변수들의 값을 보여 준다. 사용자는 목록을 만드는 지령치환을 리용할수도 있다. 다음의 for지령행은 파일 clist로부터 자기의 목록을 꺼낸다.

```
for file in `cat clist`
```

목록이 대단히 큰 경우 그리고 그의 내용들을 개별적으로 서술할수 없는 경우 이 방법은 아주 적합하다. 이것은 프로그람을 수정시키지 않고도 목록을 변경시킬수 있는 깨끗한 배렬방법이다.

그러면 이 목록은 그밖의 무엇으로 구성될수 있겠는가? 이것은 사실상 쉘이 리해하고 처리하는 임의의 표현식들로 구성될수 있다. 우리는 이미 그것이 변수와 지령치환으로써 발생된다는것을 보았다. 다음 부분에서는 목록을 통용기호와 위치파라메터들로부터 발생시킨다. for문은 아마 UNIX체계에서 가장 흔히 사용되는 순환명령문일것이다. 즉 이것을 완전히 리해하는것은 대단히 중요하다.

18.16.1 파일이름과 통용기호로 목록을 만들기

case와 같이 for문 역시 통용기호를 리용하지만 그것은 실제로 현재등록부의 파일들을 대상으로 그것들을 정합시킨다. 앞의 실례에서는 파일들의 묶음을 복사하기 위하여 for문을 사용하였다. 여기에 간단히 통용기호형태를 리용할수 있다.

```
for file in chap2[0-3] ; do
```

쉘은 자모문자순서로 분류된 파일이름목록을 만들며 순환본체의 명령들은 매 파일에 대하여 차례로 반복실행된다. 우리는 파일묶음에 대하여 작업하는 for문을 사용한 두개이상의 실례를 볼것이다.

C프로그람들의 묶음을 콤파일하기

*는 임의의 개수의 문자들과 다 정합되므로 이것을 모든 C프로그람들을 콤파일하는데 리용할수 있다.

```
$ for file in *.c ; do
> cc -o $file{x} $file                 매 목적파일이름에 첨가된 x
> done
```

cc의 -o선택항목은 출력자료로서 실행가능한 파일이름들을 선택하게 한다. 여기서는 x문자가 추가된 본래의 파일이름을 사용하고 있다. 순환은 매개 C프로그람 례하면 stringfind.cxd를 선택하여 파일 stringfind.cx을 만든다.

파일묶음에서 치환의 적용

for순환은 sed로써 파일묶음에 대하여 치환를 진행하는데서 없어서는 안되는 명령문이다. 표준출력의 방향이 개별적인 파일로 절환된 다음 같은 파일에 반대로 출력자료를 쓰기 위한 모든 처리를 for순환안에서 진행할수 있다. 현재등록부의 모든 HTML파일에 대하여 작업하는 다음의 스크립트를 실례로 보자.

```
$ cat lower2upper.sh
for file in *.htm *.html ; do
   sed 's/strong/STRONG/g
       s/img src/IMG SRC/g' $file>$$
       mv $$ $file
       compress $file
done
```

여기서 for는 매개 HTML파일을 취하며 sed로 일부 치환를 진행하고 림시변수 $$(현재쉘의 PID)에로 출력을 옮기며 다시 그것을 본래의 파일로 쓰기한다. 마지막으로 매개 파일을 압축한다.

18.16.2 위치파라메터로 목록을 만들기

for의 사용에서 가장 중요한것들중의 하나는 외부적으로 제공되는 위치파라메터들을 처리하는 능력에 있다. 다음의 스크립트 emp6.sh는 매 인수에 대하여 파일을 반복적으로 입수한다.

```
$ cat emp6.sh
for pattern in $* ; do
    grep "$pattern" emp.lst || echo "Pattern $pattern not found"
done
```

$*는 공백으로 구분되는 문자렬들의 목록으로서 인수들의 완전한 목록을 가지고 있다(18.4). 이 문자렬들은 $1, $2 등으로서 쉘에 의하여 해석된다. 우리는 개별적인 위치파라메터를 가지고 작업하지 말아야 한다. 순환은 매개 파라메터를 가지고 반복되며 변수 pattern에 그것을 할당한다. 그러면 스크립트에 네개의 인수를 넘기는것으로써 실행시켜 보자.

```
$ emp6.sh 2345 1265 4379 367
2345|james wilcox     |g.m.     |marketing |03/12/45|110000
1265|p.j. woodhouse   |manager  |sales     |09/12/63| 90000
Pattern 4379 not found
Pattern 367 not found
```

for문에 인수들을 호출하기 위하여 $*(또는 "$@")를 널리 리용한 때부터는 목록이 비게 되면 기정적으로 이 파라메터로 된다. 다음의 두 명령은 같은 의미를 가진다.

```
for pattern in $*              실제로 "$@"(18.16을 보시오)
for pattern                 기정적으로 "$@"이 놓인다
```

참고

리해하기 쉽게 하기 위해서는 스크립트들에서 <for 변수>형식보다 <for 변수 in $*>형식을 리용하는것이 좋다. 간략화는 때로는 복잡성을 가져 온다. 파라메터 $@의 리용에 대해서는 다음절을 참고하시오.

18.16.3 또 하나의 특수한 파라메터 $@

스크립트 emp6.sh는 네개의 문자로 된 종업원식별번호를 가지고 파일 emp. lst를 탐색하였다. for 순환은 식별번호대신에 이름을 사용하면 문제를 발생시킨다. 그러면 인수로서 두개의 이름을 가지고 스크립트를 실행시켜 보시오.

```
$ emp6.sh "robert dylan" "ringo lennon"
5678|robert dylan     |d.g.m.   |marketing |04/19/43| 85000
5678|robert dylan     |d.g.m.   |marketing |04/19/43| 85000
Pattern ringo not found
6213|michael lennon   |g.m.     |accounts  |06/05/62|105000
```

여기서는 흥미는 있지만 믿을수 없는 내용들이 출력되였다. for문은 robert, dylan, ringo, lennon

을 개별적인 인수로서 취급하였다. ringo는 탐색하지 못하였지만 lennon은 michael의 성이므로 탐색되였다. 여기서 한행이 두번 반복되여 나타났다는것에 주의를 돌리시오. 그러다고 하여 for pattern in "$*"와 같이 $*에 인용부호를 붙이는것은 더우기 나쁘다.

```
$ emp6.sh "robert dylan" "ringo lennon"
Pattern robert dylan ringo lennon not found
```

여기서 두개의 인수들은 "$*"에 의하여 하나의 인수로서 취급된다. 이 위험한 정황에서 벗어 나기 위한 방법은 가장 홀시되는 쉘의 특수파라메터 $@를 리용하는것이다. 이것이 인용부호없이 리용되면 파라메터 $*처럼 동작한다. 그러나 인용부호와 함께 사용되면 그것으로 묶어 진 인수들은 개별적인 인수로 취급된다. for명령문을 다음과 같이 변경시키자.

```
for file in "$@"
```

그다음 앞에서 했던것처럼 스크립트를 실행시키자.

```
$ emp6.sh "robert dylan" "ringo lennon"
5678|robert dylan     |d.g.m    |marketing |04/19/43| 85000
Pattern ringo lennon not found
```

이러한 출력은 중요한 내용을 보여 주고 있다. 여러개의 단어문자렬을 쉘스크립트의 인수로서 리용한다면 스크립트의 매개 인수에 에 대한 반복을 위하여 프로그람작성구조로서 "$@"를 리용해야 한다.

$*보다 $@를 쓰는것은 $*이 모든것을 하나의 인수로서 처리하던것과 같은 방법으로 $@가 단일한 단어인수들을 처리할수 있기때문에 널리 쓸것을 권고한다.

참고

스크립트의 인수로서 여러개의 단어문자렬을 사용해야 한다면 $*이 아니라 "$@"을 리용하시오. "$@"는 인수들을 단일단어문자렬에 대해서는 $*과 같은 기능을 수행한다. 사실상 명령문 for pattern은 실제로 for pattern in "$@"을 의미한다.

18.16.4 파일확장자의 변경(basename)

또 다른 외부지령 basename을 론의하여 보자. 이것은 for순환내에서 리용하면 대단히 효과적이다. 즉 파일묶음의 확장자를 바꾸는데 아주 쓸모가 있다. Windows사용자들은 UNIX가 다음의 지령렬을 받을수 없다는것을 알면 대단히 실망하게 될것이다.

```
mv *.txt *.doc              .txt를 가진 파일의 확장자를 .doc로 변환하려고 하는 경우
```

UNIX에서는 Windows가 한행의 RENAME지령이 하는 기능을 실현하기 위하여 스크립트를 사용해야 한다. 우리는 앞에서는 expr지령으로 기본파일이름만을 추출하였는데 basename지령도 역시 같은 처리를 하며 정규식을 리용하지 않는다.

```
$ basename /home/henry/project3/dec2bin.pl
dec2bin.pl
```

basename이 두개의 인수를 가지고 쓰일 때에는 첫번째 인수에서 두번째 인수의 문자렬을 제거한다.

```
$ basename hello.java .java
hello                  확장자 .java가 제거된다
```

이제는 파일이름의 확장자를 txt로부터 doc로 변경시키기 위하여 순환내에서 이 기능을 리용할수 있다.

```
for file in *.txt ; do
   leftname=`basename $file .txt`
   mv $file ${leftname}.doc
done
```

for문이 첫번째 파일로서 seconds.txt를 취하였다면 변수 leftname에는 문자렬 seconds가 보존된다. mv지령은 이 문자렬(seconds)에 간단히 .doc를 추가한다. 이 일감에 대해서는 expr지령이 전혀 필요 없다.

18.17 확장된 실례스크립트

앞에서 취급한 스크립트 cpback.sh를 여러개의 파일이름을 받아 들이도록 확장하는것으로써 이 장의 마지막스크립트를 개발하는 과정을 통하여 while과 for의 사용법에 대해 더 구체적으로 리해하여 보자. 이 스크립트는 적어도 두개의 인수를 사용하며 마지막인수는 반드시 등록부이여야 한다. 스크립트의 본문은 그림 18-3에 제시하였다.

인수목록으로부터 등록부이름을 제외한 두번째 목록을 구성할수 있어야 한다. 달리 말하여 마지막인수를 제외한 스크립트의 모든 인수들에 대하여 반복을 위한 for순환을 사용할수 있어야 한다. UNIX쉘은 마지막인수나 파라메터를 식별하는 아무러한 기호도 가지고 있지 않다. 그러므로 여기서는 려파기에 대한 지식을 활용해야 할것이다.

새 목록을 만들기 위하여 tr지령으로써 인수목록안에 있는 공백문자를 행바꾸기문자로 변환하였다. 이것은 매 스크립트인수들이 변수 $$에 의하여 표현되는 파일의 개별적인 행에 놓여 지도록 한다. 이 파일의 마지막행은 등록부이름(tail -1 $$)이므로 그것을 변수 destination에 기억시킨다. 그다음 인수들의 개수를 하나 덜고 그것을 변수 count에 보존한다(count='expr $# -1'). 또한 head지령을 리용하여 목록에서 마지막행을 제거하고(head -$count $$) 그것을 for순환의 목록으로 리용한다.

마지막에는 반드시 림시파일을 지워 버려야 한다(rm $$). 그밖의 모든것은 이전의것과 대부분이 같다. 그러므로 이번에는 같은 등록부인 safe에 대하여 여러개의 파일이름을 리용한 복사련습을 계속하여 보자. 우리는 같은 지령행을 여러번 반복, 실행시킨다.

```
$ cpback2.sh vvi.sh toc.pl index safe
File vvi.sh copied to vvi.sh.3
File toc.pl copied to toc.pl.1
$ cpback2.sh vvi.sh toc.pl index safe
File vvi.sh copied to vvi.sh.4
File toc.pl copied to toc.pl.2
File index copied to index.1
$ cpback2.sh vvi.sh toc.pl index safe
File vvi.sh copied to vvi.sh.5
File toc.pl copied to toc.pl.3
File index copied to index.2
```

```
#!/bin/sh
# Program cpback2.sh -- Copies multiple files to a directory
# Makes backups instead of overwriting the destination files
# Copies foo to foo.1 if foo exists of foo.2 if foo.1 exists.........

if [ $# -lt 2 ] ; then
    echo "Usage: $0 source(s) destination" ; exit
fi

echo $* | tr ' ' '\012' > $$                      #매개 인수를 개별적인 행에 놓는다
destination=`tail -l $$`                                     #마지막행은 등록부이다
if [ ! -d $destination ] ; then
    ehco "Directory $destination doesn't exist"
else
    count=`expr $# - l`                 #인수계수보다 하나 더 적은것에 대하여 반복
    for file in `head -$count $$`; do
    if[ ! -f $destination/$file ] ; then
       cp $file $destination                              #여기서는 덧쓰기되지 않는다
    else
       copies=1
       while true ; do
            if [ ! -f $destination/$file.$copies ] ; then
               cp $file $destination/$file.$copies
               echo "File $file copied to $file.$copies"
               break                                          #더이상 반복이 필요없다
            else
               copies=`expr $copies + 1`                                  #다음수에로
            fi
       done
    fi
   done
fi
rm $$                                                         #이 림시파일을 제거한다
```

그림 18-3. 덧쓰기없이 여러개의 파일을 복사하기 위한 스크립트

스크립트는 여러개의 파일이름을 가지고도 잘 동작한다. 그러면 하나이상의 파일을 복사하여도 목적지에 절대로 덧쓰기되지 않는 도구를 가지고 있다고 말할수 있겠는가?

어떤 프로그람을 변경시킬 때마다 그것을 어떤 등록부에 복사하기 위하여 스크립트 cpback2.sh를 리용할수 있으며 자동적으로 수자화된 확장자가 제공될것이다. 이 모든것에 대해서 잘 리해되지 않으면 19.5를 참고하시오.

주해

while순환에 대한 응용프로그람들의 묶음을 생각할수 있다. 실례로 기억을 되살리기 위한 배경스크립트에서 실행되게 할수 있다. 즉 vi편집기를 리용할 때 파일을 복사하기 위하여 :w지령을 쓰는 경우 화면에 이것을 쓰지 못하도록 하기 위한 경고문을 내보내기 위하여 while문을 리용할수 있다. 이것은 마지막으로 되는 중요한 련습이다.

이 장은 쉘프로그람작성을 위하여 설정한 두개의 장가운데서 첫번째 장이다. 여기서는 쉘프로그람작성의 강력한 기능을 직접 보았다. 여기서 Korn과 bash의 배타적인 기능들을 비롯하여 쉘의 개선된 기능들을 취급하기에 앞서 사용자가 알아야 할 모든것을 론의하였다. 그러면 다음장에서 새 기술을 배워 더 훌륭하고 더 간결한 코드를 개발하고 이전에 할수 없었던것들을 실현해 보자.

요 약

쉘은 한번에 한행씩 해석하는 방식으로 쉘스크립트들을 실행시키는 프로그람작성언어이다. 쉘스크립트는 C와 같은 콤파일방식의 언어들보다 실행속도가 뜨지만 대체로 일감들에 대하여 속도는 장애로 되지 않는다. 쉘변수들은 그 이름을 대괄호로 묶는것으로써 값이 평가된다. 이러한 괄호는 변수와 문자렬을 련결할 때 리용할수 있다.

쉘스크립트는 스크립트파일에 실행허락을 준 다음에야 실행된다. 그것은 또한 sh지령으로서 실행시킬수 있다. 첫행에 명령 #!/bin/sh을 놓는것으로써 스크립트가 리용할 쉘을 지적할수 있다. sh는 Korn과 bash쉘로서 작업하는 경우에는 ksh와 bash로 바꾸어 놓아야 한다. read명령은 건반으로부터 입력자료를 받아 들인다. 입력자료는 하나이상의 변수들에 읽혀 진다. read명령문을 포함하는 스크립트는 파일로부터 자료를 입력하도록 방향을 절환시킬수 있다.

지령행에 인수들을 지적하는것으로써 스크립트를 비대화식으로 실행시킬수 있다. 이 인수들은 위치파라메터들인 $1, $2 등에 넘겨 진다. 쉘파라메터 $#는 인수들의 개수를 기억하며 $*는 모든 인수들을 포함한다. $0은 실행되는 스크립트의 이름을 포함하고 있다.

모든 지령이나 스크립트는 실행완료시에 완료값(또는 돌림값)을 돌린다. 이 값은 파라메터 $?에 보존된다. 령은 참을 의미하며 령 아닌 값은 거짓을 의미한다.

&&와 ||연산자들은 하나의 분기를 가진 조건식으로 작용한다. &&가 두개의 지령사이에 쓰일 때 두번째 지령은 첫번째 지령이 성공하였을 때에만 실행된다. ||연산자에 대해서는 그와 반대이다. exit명령은 스크립트를 완료한다. 이 명령은 성공 혹은 실패를 의미하는 수값과 함께 사용될수 있다. 이 수값은 $?에 기억된다.

if명령은 세가지 형식을 가지며 fi로써 닫긴다. 해당한 명령묶음을 실행하기 위하여 조종지령의 돌림값을 평가한다. else와 elif요소들은 조종지령이 실패하는 경우 또 다른 명령묶음을 정의하는데 사용된다. 조종지령은 UNIX지령일수 있다.

test명령문은 동의어 [ ]를 가진다. 이 명령문은 수값들과 문자렬들을 비교하기 위하여 여러개의 연산자들과 함께 사용될수 있다. test는 아무런 출력도 하지 않지만 간단히 변수 $?에 자기의 검사결과를 기억시킨다.

case는 정밀한 문자정합구조이며 esac로 닫겨 진다. 이 명령은 egrep형식으로 다중패턴을 정합하는 쉘의 통용기호들을 사용한다. *가 마지막선택항목으로서 사용될 때 이것은 이전의 선택항목들에서 정합

되지 않는 모든것을 정합시킨다. 통용기호들은 파일이 아니라 문자렬들을 정합한다.

case는 파일이름 $0을 정합하는데 특별히 적합하다. 이것은 호출되는 이름에 따라서 여러가지 일을 수행하는 스크립트를 설계할수 있게 한다.

expr는 옹근수값계산과 문자렬처리에 사용된다. 이것은 Bourne쉘에서 변수값을 증가시키는데 쓰인다. 또한 부분문자렬의 추출, 문자위치의 탐색, 문자렬의 길이를 평가하기 위하여 정규식들을 사용한다.

while순환은 조종지령이 참의 값을 되돌리는한 자기의 본체부분을 반복실행한다. 이것은 반복적으로 변수값을 증가시키거나 사용자에게 여러번의 기회를 주기 위한 스크립트들에 사용된다. 조종지령으로서 true를 사용하면 무한순환을 설정할수 있다.

until순환은 while을 대신할수 있다. for는 한번 순환할 때 하나의 목록요소를 가지고 동작한다. 목록은 변수, 통용기호, 위치파라메터 또는 지령치환으로서 구성될수 있다.

또한 순환안에 코드를 놓는것으로써 C프로그람묶음을 콤파일할수 있으며 또 sed로써 여러개의 치환을 진행할수 있다.

모든 순환은 열쇠단어 do와 done을 사용한다. break명령문은 순환에서 탈퇴시키며 continue명령문은 다음의 반복에로 이행한다(순환의 다음주기를 시작한다).

for순환에서 여러개의 단어인수를 리용하는 경우에는 특수파라메터 "$@"를 사용해야 한다. 파일확장자를 바꾸기 위하여 for순환안에 basename을 사용할수 있다.

시험문제

1. 변수 x의 값이 10이라면 xx와 xx의 값은 얼마이겠는가?
2. foo.sh로서 실행된 쉘스크립트는 자기가 스크립트내부에 반영한것과는 완전히 다른것을 하였다. 왜 그렇게 되였으며 제대로 실행시키기 위해서는 어떻게 해야 하는가?
3. 독자가 Korn쉘을 리용하여 스크립트를 개발하였다면 가입쉘이 이와 다른 경우 스크립트가 Korn쉘을 리용한다는것을 어떻게 확인할수 있겠는가?
4. 스크립트가 자기자체를 호출하면 어떻게 되겠는가?
5. 스크립트가 foo -1 -t bar[1-3]로써 실행된다면 $#와 $*의 값은 무엇이겠는가?
6. 지령의 완료값이란 무엇인가? 그의 보통값은 무엇이며 값은 어디에 보존되는가.
7. 아무것도 포함하지 않는 파일이 실행되면 돌림값은 얼마이겠는가?
8. 패턴에 대하여 grep, sed, awk로 탐색하시오. 탐색에서 실패한 경우 매 지령의 돌림값을 검사해 보시오. 무엇을 알수 있는가?
9. 이 장에서 소개된 UNIX의 외부지령들은 무엇인가? 왜 여기서 그것들이 사용되였는가?
10. 스크립트를 실행시키기 위하여 sh foo.sh대신에 sh<foo.sh를 쓸수 있겠는가?

11. 프롬프트를 제시함이 없이 단번에 출력자료를 표시할수 있도록 스크립트 emp4.sh를 비대화식으로 어떻게 실행시킬수 있겠는가?
12. 인수로서 산수식들의 목록을 포함하는 파일을 받아 들이기 위한 스크립트를 쓰고 bc를 리용하여 <식=값>의 형식으로 산수식과 그 값을 표시하시오.
13. while[5]는 무엇을 하며 왜 그런가?
14. 어떤 지령을 for순환만을 써서 어떻게 10번 반복할수 있겠는가?
15. 이 프로그람의 례외적인 기능은 무엇인가?

련습문제

1. 사용자가 쉘스크립트를 실행시키지 못하도록 하자면 어떻게 해야 하는가? 스크립트파일로부터 실행허가를 제거하여야 하겠는가?
2. x의 값이 5이고 x="expr $x + 10"으로써 그것을 재할당하면 x의 새로운 값은 얼마인가? 또 외인용부호를 리용하였다면 어떻게 되였겠는가? 이 방법들은 무엇이 잘못되였는가?
3. df와 du지령들을 포함하는 test라는 이름을 가진 스크립트는 실행될 때 아무것도 표시하지 않는다. 왜 이렇게 되겠는가? 이 스크립트가 정확히 동작하도록 하기 위한 두가지 방법을 말해 보시오.
4. 홈등록부에서 인수파일의 모든 경련결을 찾기 위한 스크립트를 작성하시오. 인수로서 제공되는 파일이름은 현재등록부에 존재해야 한다.
5. lm으로 호출될 때에는 변경시간에 따라서 그리고 la로 호출될 때에는 접근시간에 따라서 파일들을 목록화하여 보여 주는 쉘스크립트를 작성하시오. 그밖에 그것을 실행하기전에 무엇을 해야 하는가?
6. 인수들로서 패턴과 파일이름을 받아 들여 파일에서 패턴의 발생회수를 계수하는 스크립트를 작성하시오(패턴은 한행에서 여러번 나타날수 있으며 영문자와 밑선기호로만 이루어 져 있다).
7. 파일들의 변경날자와 접근시간을 허락비트, 크기, 파일이름들과 함께 옆으로 나란히 보여 주는 파일들에 대한 목록을 표시하는 스크립트를 작성하시오. 스크립트는 선택적인 목록을 표시하기 위한 임의의 개수의 인수들을 받아 들여야 하며 만일 파일이름들이 인수로서 지적되지 않으면 중지되여야 한다.
8. case문을 리용하여 말단으로부터 입력된 문자렬이 최소 10개의 문자를 가지고 있지 않으면 적당한 통보문을 표시하는 스크립트를 작성하시오.
9. expr를 리용하여 우와 같은 스크립트를 작성하시오.
10. expr를 리용하여 파일의 절대경로이름으로부터 어미등록부를 추출하는 스크립트를 작성하시오.
11. 다음의 프로그람에는 적어도 6개의 오유가 있다. 그것을 찾으시오(행번호들은 왼쪽에 보여 준다).

```
1   ppprunning = yes
```

```
2  while $ppprunning = yes ; do
3  echo "    INTERNET MENU\n
4  1. Dial out
5  2. Exit
6  Choice:
7  read choice
8  case choice in
9  1) if [ -z "$ppprunning"]
10 echo "Enter your username and password"
11 else
12 chat.sh
13 endif ;
14 *) ppprunning=no
15 endcase
16 done
```

12. while순환, for순환을 리용하여 30초간격으로 체계의 프로쎄스들을 5번 표시하시오.
13. 체계에 가입된 모든 사용자들에게 통보문으로서 파일 msg.lst의 내용을 보내시오. 이때 여러번 가입한 사용자들도 단 하나의 통보문을 받게 하시오.
14. 현재등록부에서 매개 파일의 마지막 세개의 행을 head형식으로 표시하기 위한 스크립트를 작성하시오.
15. 인수로서 하나이상의 파일이름을 받아 들여 그 파일이름들을 대문자로 변환하는 스크립트를 작성하시오.
16. 두개의 등록부이름 bar1과 bar2를 받아 들여 bar2에서 bar1에서와 같은 이름을 가진 파일들을 삭제하는 스크립트를 작성하시오.
17. romeo와 henry만이 말단 tty05와 tty06에서 그것을 실행하도록 하는 스크립트를 작성하시오.
18. 분간격으로 한번씩 경고음을 울려 주고 마지막행에 통보문을 현시하도록 하는 스크립트로부터 vi를 호출하시오. 이 통보문은 반전된 영상에 표시되며 사용자가 완충기에 보관하도록 상기시킨다(참고: 유표의 위치에 맞추어 tput지령을 사용하시오. 그리고 배경에 순환을 설정하고 vi가 완료될 때 그것을 해제하시오).

제 19 장. Korn과 bash에서의 쉘프로그람작성

이 장에서는 쉘의 개선된 기능들에 대하여 본다. 또한 쉘에 의하여 마련된 환경을 분석하며 쉘흐름을 처리하기 위한 수단을 만든다. 일부 쉘명령문들은 흔히 보조쉘들에서 실행된다. 우리는 보조쉘들에로 환경파라메터들을 보내는 방법에 대하여서와 보조쉘을 기동시킴이 없이 지령들을 실행시키는 방법에 대하여 배우게 된다. 또한 쉘들에 내장된 기능들을 론의하게 된다. 이 기능들은 이전에는 외부지령으로서 존재하였다.

이 장에서는 또한 Korn쉘과 bash의 일부 중요한 기능에 대하여 론의한다. 이 두 쉘은 현대적인 수속적언어들에서의 특징들을 리용하고 있으며 수값계산, 문자렬 및 배렬을 관리할수 있다. Korn의 Ksh93판본은 쉘의 지위를 더욱 높였으며 쉘 그자체에 awk와 perl의 많은 기능들을 통합시킨 고급한 기능을 제공한다.

이 장에서 론의되는 그밖의 기능들은 Bourne쉘에서도 유효하지만 여기서 취급되는 모든 실레들을 원활하게 동작시키자면 Korn쉘과 bash의 어느 하나로 선택해야 할것이다. 또한 Korn과 bash의 공통적인 특징들을 론의하며 이러한 모든 특징들은 절의 서론부분들에서 지적된다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- set와 shift를 리용하여 단일행지령출력으로부터 마당들을 구성한다(19.1).
- here문서에 의하여 어떤 프로그람의 자료를 같은 스크립트에 배치한다(19.2).
- Korn쉘과 bash에서의 계산을 위하여 let를 리용하는 방법을 배운다(19.3).
- 순환 혹은 조건문에 대한 열쇠단어들에 방향절환을 제공하는 방법을 배운다(19.4.1).
- 기호 1>&2와 2>&1로써 표준출력흐름과 표준오유흐름을 결합한다(19.4.2, 19.4.3).
- export에 의하여 변수들이 보조쉘들에서도 유효하게 한다(19.6.1).
- 프로쎄스와 관련되는 연산자 ()와 {}의 의미를 리해한다(19.6.2).
- Korn쉘과 bash에 의하여 지원되는 배렬들을 리용한다(19.7).
- Korn쉘과 bash의 내부기능을 리용하여 문자렬들을 처리한다(19.8).
- 변수들이 설정되는가 안되는가에 따르는 각이한 방법으로 변수들을 평가한다(19.9).
- 쉘함수가 별명보다 어떻게 우월한가 하는것을 리해한다(19.10, 19.11).
- eval을 리용하여 지령행을 두번 평가하는 방법과 일반화된 프롬프트들과 변수들을 만드는 방법을 배운다(19.12, 19.13).
- exec로써 현재프로그람을 다른것으로 덧놓고 여러개의 흐름을 처리하는 방법을 배운다(19.14)
- set -x로써 쉘스크립트들에 대한 오유수정을 진행한다(19.15).
- trap를 리용하여 스크립트가 신호를 접수하는 동작을 결정한다(19.16).

19.1 위치파라메터에 값을 할당하기(set)

date와 같은 일부 UNIX지령들은 단일행출력을 만든다. 다른 지령들의 출력은 흔히 단일행을 만들어 주는 grep와 head와 같은 행추출자(line extractor)를 통하여 려과된다. 때때로 이 행에서 하나 혹은 그이상의 마당들을 호출할 필요가 있다. 우리는 앞장에서 이와 비슷한 경우를 보았다(18.10). 즉 date지령의 마당을 추출하는데 cut를 사용하였다.

```
case `date | cut -d" " -f1` in
```

단일마당을 추출하는데 외부지령을 호출하는것은 극단한 경우이다. 쉘은 이러한것을 위한 내부지령 set를 가지고 있다. 이 명령문은 대단히 간단한 기능을 수행한다. 즉 자기의 인수들을 위치파라메터들인 $1, $2 등에 할당한다. 이것은 프로그람의 출력자료로부터 개별적인 마당들을 선택하는데 특별히 유용하다. date를 지령치환으로서 리용하면 값들을 위치파라메터들에 할당할수 있다.

```
$ set `date`
$ echo $*
Thu Sep 30 08:27:50 EST 1999
$ echo "The date today is $2 $3, $6"
The date today is Sep 30, 1999
$ echo $#
6
```

set를 사용할 때 $*과 $#는 보통의 방법으로 설정된다. 이제는 임의의 마당선택에 cut를 쓸 필요가 더는 없어 졌으며 set가 그것을 쉽게 선택한다. 기정적으로 set는 공백(whitespace)으로써 문자렬값을 분해하여 그 매개를 위치파라메터들에 할당한다. Bourne쉘에서는 9개의 파라메터까지만 직접 호출가능하며 Korn과 bash에서는 임의의 파라메터에 대하여 다 호출할수 있다. 이 장의 모든 내용을 set에 설정하면 매개 단어들에 접근할수 있으며 이때 10번째 파라메터부터는 대괄호를 리용해야 한다.

```
$ set `cat ux3rdl19`
$ echo $#
20202
$ echo $4 $5
SHELL PROGRAMMING
$ echo ${12}
BASH                    bash와 ksh에서만 동작한다
```

호출할수 있는 가능성은 대단히 많다. 어떤 지령이나 관흐름의 출력자료는 아무러한 외부지령을 쓰지 않고도 set에 의하여 단어들로 분할될수 있다. set가 구분문자로서 공백(whitespace)을 사용한다는것은 쉘변수 IFS으로써 변경시킬수 있다는것을 말해 준다.

주의

스크립트가 지령행인수들을 받아 들이고 또 set명령문을 사용한다면 set를 리용하기전에 개별적인 변수들에 지령행인수들을 보관해야 한다. set는 자기자체의 파라메터에 대해서도 같은 기법을 리용하므로 인수에 의하여 리용되는 파라메터들은 덧쓰기된다.

Korn쉘

Bourne쉘과는 달리 이 쉘들에서는 직접 접근할수 있는 파라메터의 수에 제한이 없다. 실례로 40번째 파라메터에 접근하자면 다음과 같이 쓰면 된다.

```
echo ${40}
```

BASH쉘

19.1.1 set의 기정구분문자(IFS변수)

쉘변수 IFS는 지령행에서 단어분리기호로서 사용되는 문자들의 렬을 포함한다. 보통 이 문자렬은 공백, 타브, 행바꾸기문자들로 이루어 진다. 이 문자들은 보이지 않는다. 지령 echo $IFS는 빈 행을 출력한다.

```
$ echo $IFS
빈 행
```

비록 눈에는 보이지 않지만 거기에는 공백과 같은 문자들이 있다. 사용자는 8진수로써 이 변수의 값을 확인할수 있다.

```
$ echo "$IFS" | od -bc
0000000 040 011 012 012                 공백,타브,행바꾸기문자
             \t  \n  \n
0000004
```

공백은 8진ASCII값으로서 040이다. 즉 echo에서는 \t와 \n을 사용하였다. set는 자기의 경계구분문자로서 IFS의 값을 사용한다. 사용자들은 이 변수에 대하여 신경을 쓰지 않지만 한행을 어떤 다른 경계구분문자로서 분해하려면 IFS를 림시로 변경해야 한다. 실례로 파일 /etc/passwd에 대한 어떤 행을 보기로 하자.

```
$ grep henry /etc/passwd
henry:x:501:100:henry blofeld:/home/henry:/bin/ksh
```

만일 read명령으로 파일을 읽을 때 이와 같은 행을 만난다면 set를 리용하기전에 IFS의 값을 변경시켜 6번째 마당을 쉽게 추출할수 있다.

```
$ IFS=;
$ set `grep "^henry" /etc/passwd`
$ echo $6
/home/henry
```

이 장의 어떤 실례들에서는 IFS변수를 바꿀 필요가 있다. set와 IFS는 쓸모 있는 보조도구(accessory)들이지만 shift와 함께 쓰일 때에는 더 쓸모 있게 된다.

19.1.2 인수의 왼쪽밀기(shift)

많은 스크립트들은 파일이름과 같은 개별적인 실체를 지적하기 위하여 첫 인수를 사용한다. 다른 인수들은 문자렬들을 련속적인 렬로 표현하는데 그것은 대체로 파일로부터 선택되여야 할 여러가지 패턴들

이다. 우리는 목록에서 마지막인수를 어떻게 선택하는가에 대해서는 알고 있다(18.17). 그러면 이번에는 첫 인수를 없애 보자. 이러한 처리는 shift명령으로 진행한다.

shift는 위치파라메터를 그보다 하나 작은 번호가 붙은 파라메터로 이름을 바꾼다. 한번 호출되면 $2는 $1로 , $3은 $2로 등의 형식으로 바꾸어 진다. date지령으로 할당된 위치파라메터들에 대하여 shift 명령문을 리용해 보자.

```
$ echo $*
Thu Sep 30 08:27:50 EST 1999
$ echo $1 $2 $3
Thu Sep 30
$ shift
$ echo $1 $2 $3
Sep 30 08:27:50
$ shift 2
$ echo $1 $2 $3
08:27:50 EST 1999
```

왼쪽의 파라메터 $1은 shift가 호출될 때마다 매번 잃어 진다는것을 알아야 한다. 그러므로 만일 스크립트가 12개의 인수들을 리용한다면 세번 밀기하여 $9를 리용할수 있다. 이것은 인수로서 파일이름과 패턴묶음을 다 받아 들이는 스크립트를 설계할수 있도록 한다.

```
$ cat emp7.sh
# Script using the shift feature

case $# in
   0|1) echo "Usage: $0 file pattern(s)" ; exit 2 ;;
     *) flname=$1                #$1값이 잃어 지기전에 변수에 보존한다
        shift
        for pattern in "$@" ; do                # $*대신에 $@을 리용한다
             grep "$pattern" $flname || echo "Pattern $pattern not found"
        done ;;
esac
```

이렇게 하면 둘이상의 인수로서 스크립트를 리용할수 있다.

```
$ emp7.sh emp.lst
Usage: emp7.sh file pattern(s)
$ emp7.sh emp.lst wilcocks 1006 9877
3212|bill wilcocks     |d.g.m.    |accounts  |12/12/55| 85000
1006|gordon lightfoot  |director  |sales     |09/03/38|140000
```

```
Pattern 9877 not found
```

여기서 변수 flname은 문자렬 emp.lst를 보존하고 있으며 for명령문은 세개의 문자렬 wilcocks, 1006, 9897을 가지고 순환을 반복한다.

참고

shift를 사용할 때마다 제일 왼쪽에 있는 변수는 없어 지며 따라서 shift를 사용하기전에 이것을 변수에 보존시켜야 한다. 만일 네번째 파라메터에서부터 반복을 시작해야 한다면 첫 세개의 파라메터를 보관하고 그다음에 shift 3을 리용해야 한다.

19.1.3 지령치환을 방조하기(set --)

흔히 사용자들은 set명령문을 지령치환과 함께 리용한다. 특히 지령의 출력자료가 -로 시작될 때에는 문제가 제기된다.

```
$ set `ls -l unit01`
-rw-r--r--: bad option(s)
```

허가권문자렬이 -로 시작되므로(정규파일들에 대하여) set는 그것을 선택항목으로 해석하게 되며 그것은 《잘못된》 선택항목(bad option)이라고 통지한다. set에서는 인수들이 null문자렬로 평가될 때 또 다른 문제가 제기된다.

```
set `grep ppp /etc/passwd`
```

문자렬 ppp를 파일에서 찾을수 없을 때 set는 인수없이 동작하게 된다. 이 경우 기정적인 출력으로서 말단에 모든 변수를 표시한다. 이것은 사용자에게 혼돈을 가져 온다. 이 두가지 문제들을 해결하자면 set명령문다음에 두개의 이음표 --를 놓아야 한다.

```
set -- `ls -l unit01`
set -- `grep PPP /etc/passwd`
```

이렇게 하면 set는 이음표 --다음에 오는 인수들을 선택항목으로 취급하지 않는다. 두개의 이음표는 인수들이 null문자렬로 평가되는 경우 sed의 기정적인 동작을 억제시킨다.

19.1.4 set를 리용하여 빈 디스크공간을 알아보기

6.17의 df지령에 대한 출력을 보시오. 이 지령은 모든 파일체계에 대하여 현재 사용되는 디스크공간과 남은 공간을 표시한다. 그 출력의 대표적인 행은 다음과 같다.

```
/home          (/dev/dsk/c0t0d0s3 ): 107128 blocks   495356 files
```

우리는 파일체계태우기등록부(첫번째 마당)를 인수로 받아 들여 그 인수와 정합되는 행에서 여유공간을 나타내는 마당(두개의 수가운데 첫번째것)을 선택하는 스크립트를 작성할수 있다. 이 스크립트는 빈 공간의 크기가 한계값아래로 떨어 지면 통보문을 우편으로 보낸다. df.sh스크립트는 아래와 같은 처리를 수행한다.

```
$ cat df.sh
# df.sh -- Program to monitor free space on disk
```

```
# 인수로서 파일체계의 이름을 사용한다
set -- `df | grep "^$1"`
if [ $4 -lt 200000 ] ; then
      echo "Free space in $1 has dropped to $4 blocks" | mail root
fi
```

이 UNIX체계에서 파일체계는 첫번째 렬에 나타나며 이러한것으로 하여 grep지령에서 패턴을 고정시키기 위한 문자 ^를 사용하였다. 수값비교는 4번째 마당에 대하여 진행되였다. 통보문은 그 파일체계에서 빈 공간이 200000블로크아래로 떨어 질 때 뿌리로 전송된다. 스크립트는 태우기등록부(mounting directory)의 이름과 함께 아래와 같이 호출된다.

```
df.sh /home                /home파일체계의 빈 공간 탐색
```

이것은 체계관리자가 흔히 반복적으로 실행시키는 스크립트이므로 crontab파일에 놓는것이 가장 좋다. 즉 아래와 같이 crontab파일에 기입하면 매일 오전 9시부터 오후 5시까지사이에 시간간격으로 스크립트가 실행된다.

```
0 09-17 * * 1-5 /home/admin/scripts/df.sh
```

19.2 here문서(<<)

프로그람이 읽어 들이는 자료가 고정되여 있는 경우가 있다. 쉘은 스크립트를 포함하는 같은 파일로부터 자료를 읽어 들이기 위하여 기호 <<를 제공한다. 이것을 **here문서**라고 말하는데 자료가 개별적인 파일에 있는것이 아니라 그자체에 직접 있다는것을 의미한다. 표준입력을 사용하는 임의의 지령은 here문서로부터 입력자료를 취하게 할수 있다.

이 기능은 파일이름을 인수로서 받아 들이지 않는 지령(례를 들어 mail지령과 같이)에서 유용하다. 통보문이 짧다면(mail통보문들이 대체로 그렇다.) 같은 스크립트안에 지령과 통보문을 다 놓을수 있다.

```
mail juliet << MARK
Your program for printing the invoices has been executed
on `date`. Check the print queue            지령치환이 가능하다
The updated file is known as $flname
MARK
```

here문서기호(<<)뒤로는 세 행의 자료와 경계구분문자(문자렬 MARK)가 놓인다. 쉘은 MARK에 의하여 제한되는 모든 행을 지령의 입력으로서 취급한다. juliet는 다른 쪽에서 세개 행의 통보문을 볼것이다. 그때 MARK라는 단어는 나타나지 않는다. 이 단어가 스크립트안에 놓이면 mail지령이 외부파일을 읽어 들이는것이 없으므로 실행속도는 더 빨라 진다.

주해 here문서의 내용들은 그것이 지령의 입력자료로서 리용되기전에 쉘에 의하여 해석, 처리된다. 이것은 입력자료에 지령치환이나 변수들을 리용할수 있다는것을 의미한다. 그러나 보통의 표준입력으로서는 그렇게 할수 없다.

대화형프로그람들에서 here문서의 리용

많은 지령들은 사용자로부터 입력자료를 요구한다. 흔히 지령에 의해 제기되는 련속적인 질문들에 대한 응답으로서 같은 건이 눌리우게 된다. 실례로 지령이 림시로 정지되였을 때 y를 두번 또는 세번 눌러야 한다. 사용자가 질문에 대기하는것이 아니라 here문서로부터 입력자료를 가지도록 스크립트에 지적할수 있다.

Linux의 fdisk지령을 생각해 보자. 이 지령은 내부지령으로서 파일구획(partition)을 보여 주는 p지령과 fdisk에서 탈퇴하는 q지령을 리용한다. 체계관리자로서 자기의 과제를 처리할 때 자기가 하드디스크구획들에 대한 정확한 장치이름들을 사용하는가를 확인해야 할 필요성이 있다. 이러한 검사는 아래의 스크립트로서 처리할수 있다.

```
# cat showpart.sh                    뿌리프롬프트 #를 리용한다
fdisk << END
p
q
END
```

뿌리프롬프트로부터 showpart.sh를 실행시켜 구획정보를 즉시에 얻을수 있다. 앞에서 이것을 한번 리용하여 보았다. 여기서 우리는 대화형프로그람을 비대화형으로 동작하게 하였다. fdisk의 대표적인 출력을 21.9에 보여 준다.

참고

만일 하나이상의 read명령문과 미리 정의된 응답묶음을 가지고 실행되는 스크립트를 작성한다면 그 스크립트를 비대화적으로 실행시키는데 here문서를 리용할수 있다. 이것은 자동화에 도움을 준다.

19.3 계산(let)

Korn과 bash는 외부지령 expr와 같은 옹근수처리기능을 내장하고 있다. let명령문으로도 수값계산을 진행할수 있다.

```
$ let sum=sum=256+128                 변수다음에 공백이 없음
$ echo $sum
384
```

읽기 편리하게 하기 위하여 공백을 리용한다면 반드시 식을 인용부호로 묶어야 한다.

```
$ let sum="3 * 6 + 4 / 2" ; echo $sum
20
```

그러면 let지령이 어떻게 변수들을 처리하는가를 보자. 먼저 세개의 변수를 정의하시오. let는 다음과 같이 처리한다.

```
$ let x=12 y=18 z=5
$ let z=x+y+$z                 let에서 $는 필요 없다
```

```
$ echo $z
35
```

let지령으로 변수에 값주기를 진행할 때 기호 $를 리용하지 않아도 된다. 이 연산기능은 내장되여 있으므로 스크립트들은 expr로서 사용될 때보다 더 빨리 실행된다. let에 의한 연산은 아직은 옹근수만으로 제한된다. 그러나 Korn쉘의 Ksh93판본에서는 류동소수점연산이 가능하다. 또한 printf문이 이 두 쉘들에 내장되였으므로 awk를 사용하지 않고도 많은것을 할수 있다.

((와))로서 계산을 진행하기 위한 두번째형식

Korn쉘과 bash쉘에서는 let명령을 대신하는 (( ))연산자를 리용할수 있다.

```
$ x=22 y=28 z=5
$ z=$((x+y + z))              공백은 그리 중요하지 않다
$ echo $z
55
$ z=$((z+1))                z=$((z+=1))를 리용할수도 있다
$ echo $z
56
```

POSIX명세는 let보다도 ((와))를 사용할것을 요구하며 이 형식은 쉘들의 표준기능으로 될것 같다. 또한 변수앞에 $을 리용하지 않으므로 사용하기가 보다 더 쉽다. 그러나 전체적인 산수연산식에 대해서는 그앞에 $를 붙여야 한다.

19.4 방향절환

우리는 아직 방향절환(redirection)에 대하여 다 고찰하지 않았다. 지금까지는 가장 단순한 방법으로 절환을 리용하여 왔다. 또한 매개의 흐름들을 결합하지 않고 개별적인 흐름으로서 취급하여 왔다. 여기서는 이것을 리용한 보다 더 효과적인 방법에 대하여 배우게 된다. 또한 두개의 서로 다른 흐름들을 결합하는 몇개의 새로운 기호들을 사용하게 된다.

19.4.1 열쇠단어에 의한 방향절환

기호 >>로써 파일 newlist에 행을 추가하는 스크립트 emp5.sh(18.14)을 보기로 하자.

```
echo "$code|$description" >> newlist
```

우의 지령에서는 echo가 호출될 때마다 파일 newlist이 열린다. 이것은 작업전반에 걸쳐 효과적이지 못하다. 그러나 열쇠단어 done에 방향절환기능을 제공하는것으로써 이와 같은 여러번의 파일열기닫기를 피할수 있다.

```
done>newlist
```

여기서 파일은 한번만 열리고 닫겨 지지만 매우 중대한 결과를 발생시킨다. 즉 표준출력을 사용하는

순환내에서의 모든 지령들은 파일 newlist로 절환된다. 그러나 어떤 지령이 반드시 말단에 대하여 출력을 가져야 한다면 그때에는 /dev/tty로 방향절환을 시켜야 한다. 아래에 프로그람 emp.sh의 개정판을 주었다.

```
$ cat emp5a.sh
answer=y                #y가 순환에 먼저 들어 가도록 설정한다
while [ "$answer" = "y" ]               #조종지령
do
     echo "Enter the code and description:\c">/dev/tty
     read code description               #둘 다 읽는다
     echo "$code|$description"                #여기에는 방향절환이 없다
     echo "Enter any more (y/n)? \c" >/dev/tty
     read anymore
     case $anymore in
       y*|Y*) answer=y ;;               #y 혹은 Y로 시작하는 어떤것
       n*|N*) answer=n ;;               #n 혹은 N으로 시작하는 어떤것
           *) answer=y ;;               #다른 응답들을 y로 된다
     esac
done>newlist                  #하나의 echo명령문만이 파일에 간다
```

이 스크립트에 /dev/tty에로의 방향을 가리키는 두개의 명령이 있다. done열쇠단어에서의 방향절환이 순환내의 모든 지령들의 표준출력을 파일 newlist로 절환시키지만 이 두 명령문의 출력은 명백히 말단으로 절환된다.

방향절환은 fi나 esac열쇠단어들에서도 쓸수 있으며 입력절환이나 관흐름처리에서도 가능하다.

```
fi > foo                if와 fi사이의 모든 명령문들
esac > foo              case와 esac사이의 모든 명령문들
done < param.lst        parm.lst로부터 입력을 가진다
done | while true       while순환에로 출력을 관련결한다
```

앞장에서 마지막형식을 은밀히 사용하였다. 즉 여기서는 그것을 리용하는 규칙에 대하여 보았다.

19.4.2 흐름의 결합(1>&2)

/dev/tty에 의한 방향절환은 문제에 따라 매번 계속 발생되여야 한다. 일단 한번 절환된 흐름은 다시 방향절환될수 없다. 이것은 말단으로 출력하고 싶을 때마다 매번 >/dev/tty라는 9개의 문자를 입력해 넣어야 한다는것을 의미한다. 쉘은 이것을 더 간단하게 하는 흐름결합(stream merging)기능을 제공한다.

생각은 간단하다. 표준오유흐름은 말단으로 가므로 출력을 말단으로 정한 명령문들은 그것들의 표준출력을 이 오유흐름과 결합시켜 방향절환시킬수 있다. 이 명령문을 조종하기 위하여서는 표준오유흐름만을 처리해야 한다. 결합문자로서 &연산자를 절환기호와 함께 사용한다. 스크립트에서 echo명령문을 다음과 같이 사용할수 있다.

```
echo "None of the patterns found" 1>&2
```

이것은 《두 흐름을 결합하여 표준오유에 대한 목적지(기정적으로 말단)에로 그 결합된 흐름을 보낸다》라고 말할수 있다. 만일 순환을 절환한다면(혹은 전체 스크립트를 개별적인 파일에로 절환한다면) 이 명령문에 대한 출력은 언제나 말단에서만 볼수 있을것이다. 1이 표준출력의 기정적인 파일서술자이므로 >&2를 쓸수 있다.

주해

결합된 흐름은 보통의 방식으로 절환시킬수 있다. find_number.sh> foo 2>bar를 리용하면 기호렬 1>&2를 가진 모든 스크립트명령문들은 실제로 bar에 씌여 지게 된다. 또한 스크립트출력의 결과는 foo에 보존되여 있을것이다.

19.4.3 오유통보문과 출력을 같은 파일에 보관하기(2>&1)

표준출력과 표준오유흐름의 역할을 바꿀수 있다. 다음의 지령은 여느때와 같이 두개의 흐름을 결합하지만 이때 표준오유흐름은 표준출력방향으로 간다.

```
cat foo1 foo2 foo3 > bar 2>&1
```

그러면 1>&2와 2>&1사이에 차이가 있는가? 있다. 이 실례에서 foo3파일이 열리지 않는다면 cat의 오유통보문은 파일 foo1과 foo2의 내용들과 함께 bar에 보관된다.

만일 이러한 내용을 리해하였다면 우의 명령문은 아래와 같이 표현될수 있다.

```
cat foo1 foo2 foo3 2> bar 1>&2
```

이것은 스크립트개발자나 체계관리자에게 있어서 대단히 중요한 내용을 담고 있다. 만일 자기가 없을 때 어떤 프로그람을 실행시켜야 한다면 스크립트를 절환시켜 출력과 오유통보문들을 다 같은 파일에 보관시킬수 있다.

```
content.sh 2>&1 | sort > bar              결합된 출력을 정렬시킨다
content.sh 2> bar 1>&2
```

bar는 두 흐름에 대한 출력을 다 포함한다. 첫번째 형식에서는 흐름을 다른 지령으로 관련결시킨다. 스크립트를 실행시키는데 cron을 리용한다면 이것은 자기의 프로그람에서 틀린것을 알아 내는 유일한 방법이다.

주해

표준출력을 사용하는 지령에 기호렬 1>&2나 2>&1을 붙이면 표준출력과 표준오유의 흐름을 결합하기만 하며 그것들을 방향절환시키지는 않는다. 만일 결합된 흐름의 방향을 절환하고 싶다면 절환기호를 추가적으로 주어야 한다.

19.5 실례스크립트

그러면 /etc/passwd와 /etc/group로부터 사용자의 세부정보를 표시하는 순환절환과 흐름결합에 대한 흥미 있는 응용프로그람을 보기로 하자. Korn쉘스크립트는 두개의 인수로서 UID에 대한 범위를 받아 들이며 /etc/passwd로부터 매개의 UID에 관계되는 마당들에서 읽는다. 또한 GUID값에 대하여 /etc/group을 보고 그룹이름을 선택한다.

스크립트는 또한 대부분의 UNIX체계들에서 유효한 printf지령을 리용한다. 이것은 bash와 ksh93에

서는 내부지령이다. 이 printf에 대하여 새로운것은 아무것도 없다. 즉 이 지령은 출력을 양식화하기 위하여 awk에서 리용하였다. 스크립트 user_de tails.sh를 그림 19-1에 보여 준다. 이 스크립트에서 print 지령은 점을 사용하지 않는다. 이 모든것을 기억해 둘 필요가 있다. 두개의 인수들은 변수 uidmin와 uidmax에 보관되며 while순환은 /etc/passwd의 매행을 차례로 읽는다. 매행은 read명령에서 볼수 있는바와 같이 일곱개의 마당으로 분할된다. 네번째 마당(gid)은 set명령으로서 /etc/group에서 탐색된다. 변수 Kount는 /etc/passwd로부터 읽혀 지는 행의 수를 계수한다.

```
#!/bin/ksh
case $# in
  2) ;;                                                        # 두 인수가 요구된다
  *) echo "Usage: $0 min_guid max_guid" 1>&2 ; exit
esac

IFS=:
uidmin=$1 ; uidmax=$2                     #그것들을 설정하기 전에 $1과 $2를 보관한다
echo "\
Username    UID  GUID  Gname   GCOS        Home Directory     Login Shell
-------------------------------        ---------------------------------"
kount=0
while read username password uid gid gcos homedir shell
do
    if [ $uid -ge $uidmin -a $uid -le $uidmax ] ; then
          set -- `grep ":$gid:" /etc/group`              #이전의 $1과 $2를 돌려 준다
          gname=$1
          printf "%-10s %4d %4d   %-8s %-14s %-14s     %-12s\n"\
               $username $uid $gid $gname $gcos $homedir $shell
          kount=`expr $kount + 1`
    fi
done < /etc/passwd
case $kount in
    0) echo "No lines found" 1>&2 ;;
    *) echo "$kount lines found" 1>&2
esac
```

그림 19-1. user_details.sh스크립트

이 순환은 done열쇠단어에서 입력자료를 가진다. 표준출력흐름과 표준오유흐름을 결합한(1>&2에 의하여) 세개의 명령을 잘 고찰해 보시오. 이것은 전체 스크립트를 안전하게 파일로 절환시킬수 있으며 말단에 통보문들을 가진다는것을 의미한다. 다음과 같이 세가지 방법으로 스크립트를 실행시켜 보자.

```
$ user_details.sh > newlist                인수 없음
Usage: user_details.sh min_guid max_guid
$ userdetails.sh 701 703 > newlist              유효하지 않는 인수들
No lines found
```

```
$ userdetails.sh 601 603 > newlist              유효한 인수들
3 lines found
```

흐름결합은 잘된것 같다. 그러면 newlist의 내용을 보기로 하자.

```
$ cat newlist
Username   UID  GUID    Gname  GCOS            Home Directory  Login Shell
--------------------------------------------------------------------------
romeo      601  100     users   Romeo N.       /home/romeo      /bin/ksh

uliet      602  100     users                  /home/juliet     /bin/bash
henry      603  100     users   henry blofeld  /home/kaust      /bin/bash
```

두 명령의 표준출력(echo와 print면 newlist f)이 어떻게 이 파일로 절환되였는가를 보시오. 여기서는 두개의 파일의 내용을 읽은 다음 printf문으로써 양식화된 출력을 산출하였다. 또한 오직 한개의 외부지령 grep을 사용하였다.

주해

스크립트를 Bourne쉘에서 실행시킨다면 두개의 외부지령(grep와 printf)을 사용하여야 한다. 그러나 여기서 파일 newlist의 해당한 내용들을 보았다 하여도 No lines found통보문을 받게 된다. Bourne에서의 while순환은 보조쉘에서 실행되며 따라서 순환내에서 Kount의 값은 그의 밖에서는 쓸수 없다. 이 문제를 해결하는것이 exec명령이다(19.14).

19.6 부분쉘에서의 문제점

프로세스가 쉘에 의하여 만들어 질 때 쉘은 자기 환경의 일정한 기능들을 새끼프로쎄스에서 유효하게 만든다. 만들어 진 프로쎄스(지령을 의미한다.)는 아래와 같은 계승된 파라메터들을 자기의 연산에 리용할수 있다. 파라메터들은 다음과 같다.

- 어미프로쎄스의 PID
- 프로쎄스의 UID와 GUID
- 현재작업등록부
- 세개의 표준파일들
- 어미프로쎄스에서 유효한 일부 환경변수들

우리는 쉘스크립트들내(보조쉘에서 실행되는)에서 변수를 정의하지 않고도 HOME, SHEEL과 같은 변수들은 리용하였다. 그러나 .profile이나 다른 스크립트들에서 정의한 변수들은 어떻게 되겠는가? 보조쉘에서 그것들을 리용할수 있는가? 우리가 보아야 할 쉘환경과 관련되는 몇개의 개념들이 있는데 이것들을 다음의 단락들에서 취급한다.

19.6.1 쉘변수들을 반출하기

기정적으로 쉘변수들에 보존된 값들은 새끼쉘에 넘겨 지지 않는다. 그러나 쉘은 전역적으로 유효하도록 이 변수들을 모든 새끼프로쎄스들에 반출(export)할수 있다(export명령을 써서). 앞에서는 다만 이 명령문을 사용하는데 그쳤지만 여기서는 왜 그렇게 하였는지 리해할것이다.

그러면 스크립트가 실행되기전에 쉘에서 정의되는 변수 x의 값을 표시하는 간단한 스크립트를 생각해 보자.

```
$ cat var.sh
echo The value of x is $x
x=20                #x의 값을 변화시킨다
echo The new value of x is $x
```

먼저 프롬프트에서 변수 x에 값 10을 할당하고 그다음 스크립트를 실행시킨다.

```
$ x=10 ; var.sh
The value of x is                x의 값은 보조쉘에 존재하지 않는다
The new value of x is 20
$ echo $x              스크립트내에서의 값설정은 스크립트밖의 값에 영향을 미치지 않는다
10
```

x는 가입쉘에서만 쓸수 있는 국부변수이므로 그의 값은 보조쉘에서 실행되는 스크립트에서 echo에 의해 표시할수 없다. x를 전역적으로 유효화하자면 스크립트가 실행되기전에 export명령문을 사용해야 한다.

```
$ x=10 ; export x
$ var.sh
The value of x is 10             스크립트밖에서의 변수설정은 여기서 볼수 있다
The new value of x is 20
$ echo $x
10                 스크립트내에서 재설정된 값은 스크립트밖에서 보이지 않는다
```

x가 반출되면 그의 값(10)은 스크립트에서 유효하다. 그러나 변수를 반출시키면 스크립트에서 변수의 재할당값(x=20)은 그 스크립트를 실행시킨 어미쉘에서는 볼수 없다.

변수들에 대해서 그의 값들을 보조쉘에로 계승시켜야 한다면 정의한 변수들에 대하여 반드시 반출시켜야 한다. 이미전에 반출된 변수들을 확인하자면 export지령을 인수없이 리용하시오. 이렇게 하면 이미 반출된 환경변수들과 사용자정의변수들을 목록형식으로 보여 준다. env지령도 반출된 변수들을 보여 준다.

주해

변수는 그것이 정의된 프로쎄스에 대하여 국부적이지만 그 변수들을 반출시키면 모든 새끼프로쎄스들에서 재귀적으로 쓸수 있다. 그러나 새끼프로쎄스가 그 변수의 값을 변경시키면 그 변화가 어미에게까지 전달되지는 않는다. 많은 사람들이 변수값을 변경시킬 때마다 매번 export지령을 사용하는 실수를 범한다. 실제로 그것은 필요 없다.

19.6.2 지령의 묶기

쉘은 지령들을 묶는데 괄호 ()를 쓰지 않고 {}를 리용한다. 둘사이의 차이점을 본다면 앞의것은 보조쉘에서 지령묶음을 실행시키지만 다른것은 현재쉘에서만 사용한다. 이것은 내부지령 cd와 pwd지령과 함께 두 연산자를 리용해 보면 명백히 알수 있다.

```
$ pwd
```

```
/home/romeo
$ ( cd progs ; pwd )
/home/romeo/progs
$ pwd
/home/romeo                    본래의 등록부로 돌아 간다
```

보조쉘의 기동과 동시에 cd지령은(환경파라메터중의 하나) 작업등록부를 /nome/romeo/progs로 바꾸었다. 어미(가입쉘)는 이 변화를 접수할수 없으므로 본래등록부가 되돌려 진다. 이번에는 {}연산자를 써서 같은 지령묶음을 만들어 사용하여 보자.

```
$ pwd
/home/romeo
$ { cd progs ; pwd
>}
/home/romeo/progs
$ pwd
/home/romeo/progs              등록부변경이 허락된다
```

두개의 지령은 보조쉘을 기동시키지 않고 실행되며 등록부는 변화된채로 남아 있다.

참고

대괄호는 제각기 다른 행에 놓인다. 만일 두개가 다 한행에 놓일것을 원한다면 마지막지령다음에 반두점을 놓아야 한다.

```
{ cd progs ; pwd ; }
```

그러면 이러한 지령묶기기능은 어디에 필요하겠는가? 제18장에서 개발하였던 스크립트를 다시 보기로 하자. 거기서 우리는 일정한 부분을 지령묶음으로써 바꿀수 있다는것을 알게 될것이다. 실례로 스크립트 emp3b.sh(18.9)의 시작부분을 간단히 아래와 같이 바꿀수 있다.

```
[ $# -ne l ] && { echo "Usage: $0 pattern" ; exit 3 ; }
```

여기서 프로그람은 사용자가 인수를 지적하지 않으면 포기된다.

주해

()연산자들은 보조쉘에서 그것으로 둘러 막힌 지령들을 실행시키며 {}연산자들은 현재쉘내에서 그것을 한다. 이것은 지령묶음내에 exit명령이 있는 경우 {}를 리용해야 한다는것을 의미한다.

19.7 배렬

Korn과 bash는 첨수 0으로 시작하는 1차원배렬을 지원한다. Ksh93에서는 요소개수가 4096개로 제한되지만 bash에서는 이러한 제한이 없다. 그러면 배렬 prompt의 세번째 요소를 어떻게 설정하며 평가하는가를 보기로 하자.

```
$ prompt[2]="Enter your name"
$ echo ${prompt[2]}
Enter your name
```

값의 평가는 {}로 진행하며 prompt[2]는 변수처럼 취급된다. 이 배렬변수는 같은 쉘에서 정의되는 변수 prompt와 충돌하지 않는다. 요소묶음에 대하여 값을 할당하려면 괄호로서 닫겨 진 공백으로 구분되는 목록을 사용할수 있다.

```
month_arr=(0 31 29 31 30 31 30 31 30 31 31 30 31)
```

이것은 배렬 month_arr를 정의한다. 이 문법적구조는 bash와 Ksh93에서 사용된다. 만일 Korn의 낡은 판본을 쓰고 있다면 set -A명령문을 리용할수 있다.

```
set -A month_arr 0 31 29 31 30 31 30 31 30 31 31 30 31
```

어느 경우에나 배렬은 요소별로 매달의 유효한 일수를 보관한다. perl의 배렬들은 첫번째 대입형식을 사용한다. 첫 요소는 명백한 판단을 위하여 0을 의도적으로 대입하였다. 6월달의 유효한 일수를 알아내는것은 간단하다.

```
$ echo ${month_arr[6]}
30
```

보조스크립트로서 @ 혹은 *을 리용하면 요소들의 개수는 물론 배렬의 모든 요소들을 표시할수도 있다. 아래의 형식들은 요소수를 표시하기 위하여 기호 #을 리용한다는 점을 제외하고는 류사하다.

```
$ echo ${month_arr[@]}
0 31 29 31 30 31 30 31 30 31 31 30 31
$ echo ${#month_arr[@]}              배렬의 길이
13
```

그러면 배렬을 리용하여 입력된 날자의 유효성을 검사할수 있는가? 그림 19-2에 보여 주는 스크립트 dateual.sh는 그러한 검사를 진행하며 윤년(400년을 주기로 진행되는것을 제외하고는)들에 대한 2월달의 일수변화를 고려한다.

case구조의 첫 선택항목은 null응답에 대하여 검사한다. 두번째 선택항목은 mm/dd/yy형식으로 되여 있는 8개의 문자로 이루어 진 문자렬을 검사하기 위하여 표현식 $n/$n/$n을 사용한다.

IFS의 변경된 값을 사용하여 날자의 구성요소들은 세개의 위치파라메터에 설정되며 그것이 유효한 달인가를 검사한다. 두번째 case구조는 윤년에 대하여 검사하며 배렬을 리용하여 날자의 유효성검사를 진행한다. continue명령문은 검사가 실패할 때마다 순환을 새로 시작하게 한다.

그러면 스크립트를 검사해 보자.

```
$ dateval.sh
Enter a date: [Enter]
No value entered
Enter a date: 13/28/00
Illegal month
```

```
Enter a date: 04/31/00
Illegal day
Enter a date: 02/29/01
2001 is not a leap year
Enter a date: 02/29/00
[Ctrl-c]
```

우에서 보는바와 같이 이 스크립트는 완료경로가 명백치 않은것으로 하여 새치기건을 사용하여 실행을 완료시켜야 하였다. 우리는 이러한것을 조종하기 위한 방법을 론의할것이다.

```
#!/bin/ksh
IFS="/"
n= "[0-9][0-9]
set -A month_arr 0 31 29 31 30 31 30 31 30 31 31 30 31

while echo "Enter a date: \c" ; do
   read value
   case "$value" in
         "") echo "No value entered" ; continue ;;
   $n/$n/$n) set $value
              let rem="$3 % 4"                                   # 윤년에 대한 검사
              if [ $1 -gt 12 -o $1 -eq 0 ] ; then
                echo "Illegal month" ; continue
              else
                case "$value" in
               02/29/??) [ $rem -gt 0] &&        # 2월달이 29일까지인가 28일까지인가?
                            { echo "20$3 is not a leap year" ; continue; } ;;
                      *) [ $2 -gt ${month_arr[$1]} -o $2 -eq 0 ] &&
                            { echo "Illegal day" ; continue ; } ;;
                esac
          *) echo "Invalid date" ; continue ;;
   esac
   echo "$1/$2/$3" is a valid date
done
```

그림 19-2. 배렬을 리용한 날자의 유효성검사스크립트(dateval.sh)

19.8 문자렬처리

Korn과 bash에서는 자체내부에 충분한 문자렬처리기능들이 구비되여 있기때문에 expr외부지령이 필요 없다. expr와 달리 여기서는 정규식이 아니라 통용기호를 리용한다. 사용되는 모든 형식들은 몇개의 특수기호들과 함께 변수이름을 대괄호로 둘러 막을것을 요구한다. 그러한 형식들에서 미묘한 차이는

그것들을 기억하기 힘들게 하며 작업하기 불편하게 만든다.

문자렬의 길이

문자렬의 길이는 변수이름앞에 #를 붙여 쉽게 알아 낼수 있다. 다음의 실례를 보자.

```
$ name="vinton cerf"
$ echo ${#name}
11
```

앞에서 문자렬길이계산에 expr를 사용하였는데(18.11) 여기서의 내부기능은 사용하기가 훨씬 더 쉽다.

```
if [ `expr "$name" : '.*'` -gt 20 ] ; then          expr를 리용한다
if [ ${#name} -gt 20 ] ; then                   Korn과 bash
```

두번째 형식은 훨씬 읽기 편리하며 외부지령을 호출하지 않기때문에 속도가 더 빠르다. perl은 배렬의 길이를 평가하기 위한 류사한 형식을 사용하므로 결국 기억하기도 쉽다(20.8).

부분문자렬의 추출

ksh93과 bash는 부분문자렬을 추출하는 간단한 기술을 제공한다. 같은 변수의 값을 리용하여 awk와 perl에서 사용하였던 substr()함수의 기능을 실현해 보자.

```
$ echo ${name:3:3}                첫 위치는 령이다
ton
$ echo ${name:7}                  문자렬의 나머지부분을 추출한다
cerf
```

이 기능은 외부지령 expr에 의해 사용되였던 정규식지향기술보다 훨씬 더 사용하기 쉽다.

패턴정합에 의한 문자렬의 추출

특수한 패턴정합기술을 리용하여 부분문자렬을 추출할수 있다. 이 함수들은 두개의 문자 #와 %를 사용한다. #와 %는 각각 시작부분과 끝부분에서 정합을 진행하기 위하여 리용되며 변수의 값을 평가할 때에는 대괄호안에서 리용된다.

앞에서는 파일이름의 확장자를 제거하기 위하여 외부지령 basename(18.16)을 리용해야 하였다. 여기서는 변수의 ${변수 %패턴}형식을 리용하여 추출을 진행할수 있다. 아래에 이러한 형식의 리용방법에 대하여 보여 준다.

```
$ filename=qoutation.txt
$ echo ${filename%txt}
quotation                 txt가 제거되였다
```

변수이름다음에 오는 기호 %는 끝부분에서 변수의 내용을 정합시키는 가장 짧은 문자렬을 삭제한다. 대신에 %를 두번 쓰면 가장 긴 문자렬을 정합할수 있다. FQDN에서 주콤퓨터이름을 추출하자면 통용기호와 함께 리용해야 한다.

```
$ fqdn=java.sun.com
$ echo ${fqdn%%.*}
```

```
java
```

basename이 경로이름에서 기본파일이름만을 추출할수도 있다는것을 다시 생각해 보자. 이것은 변수값의 시작위치에서 패턴 */와 정합되는 가장 긴 패턴을 제거할것을 요구한다.

```
$ filename="/var/spool/mail/henry"
$ echo ${filename##*/
henry
```

우의 스크립트는 패턴 */와 정합되는 가장 긴 패턴 즉 토막 /var/spool/mai/을 제거한다. 정합규칙을 알고 있으므로 한개의 #를 가지고 정합해 볼수 있다. Korn과 bash의 패턴정합형식들을 표 19-1에 보여 준다.

표 19-1. bash와 ksh에 대한 패턴정합연산자

형 식	제거한후에 남아 있는 토막에 대한 평가
${var#pat}	$var의 시작부분에서 pat와 정합되는 가장 짧은 토막
${var##pat}	$var의 시작부분에서 pat와 정합되는 가장 긴 토막
${var%pat}	$var의 끝부분에서 pat와 정합되는 가장 짧은 토막
${var%%pat}	$var의 끝부분에서 pat와 정합되는 가장 긴 토막

19.9 조건부파라메터치환

변수평가에 대한 론의를 계속하여 보자. 변수가 null값을 가졌는가 혹은 어떤 정의된 값을 가졌는가에 따라서 변수를 평가할수 있다. 여기서도 역시 대괄호를 리용한다. 그밖에도 변수이름뒤에 두점 :을 놓아야 하며 그다음에 +, -, =, ?와 같은 기호들을 리용해야 한다. 기호의 뒤에는 문자렬이 온다. 이 기능을 **파라메터치환**(parameter substitution)이라고 하며 Bourne쉘에서도 유효하다.

+선택항목

형식은 ${변수:+문자렬}이다. 여기서 변수는 그의 값이 null이 아닐 때 문자렬로 평가된다. 이것은 등록부가 비지 않았을 때 통보문을 표시하기 위한 가장 좋은 방법이다.

```
found=`ls`
echo ${found:+ "This directory is not empty"}
```

ls지령은 파일들을 찾지 못하면 아무것도 표시하지 않으며 이 경우에 변수 found는 null문자렬로 설정된다. 그러나 ls가 적어도 한개 파일이라도 발견하면 통보문이 표시된다.

-선택항목

+와 반대의 기능을 수행한다. 이 선택항목은 파일이름에 대하여 질문할 때 사용자가 [Enter]를 누르면 기정적인 값을 사용하는 스크립트에서 사용된다.

```
echo "Enter the filename : \c"
read flname
fname=${flname:-emp.lst}
```

프롬프트에 아무것도 입력하지 않으면 변수 flname은 emp.lst로 평가된다. 그러나 fname은 실제로 이 값을 포함한다. 여기서 대입에는 if조건문이 필요 없다.

=선택항목

이것은 앞으로 한번만 처리가 진행되며 평가되는 변수에 대입을 진행한다는것을 제외하고는 비슷하게 동작한다. 즉 x=1; while [$x -1e 10]와 같이 순환변수를 초기화하고 검사하기 위한 개별적인 명령문을 리용하는것이 아니라 while[${xx:=1} -1e 10]과 같이 단 하나의 명령문에 그것들을 포함시킨다.

?선택항목

이것은 값이 null인 경우 쉘을 중지하거나 제거한다는것을 제외하고는 -선택항목과 비슷하게 동작한다. 사용자가 응답에 실패하면 스크립트를 완료할수 있다.

```
echo "Enter the filename : \c"
read flname
grep $pattern ${flname:?"No filename entered .... quitting"}
```

=를 제외한 이 모든 연산자들은 위치파라메터들과 함께 사용될수 있다. 다음장에서 초기의 스크립트 지령렬의 일부를 압축하기 위하여 이 지식을 리용할것이다. 쉘의 파라메터치환기능들은 표 19-2에 보여준다.

표 19-2. 파라메터치환연산자

형 식	값평가
${var:+pat}	변수가 설정되지 않으면 pat, 그렇지 않으면 null
${var:-pat}	변수가 설정되면 $var, 그렇지 않으면 pat
${var:=pat}	우에서와 같다. 그러나 var를 pat로 설정한다.
${var:?pat}	var가 설정되면 $var를 그렇지 않으면 pat를 출력하며 우와 같다.

주해

이 두개의 절에서 론의되는 문자렬처리기술은 단순히 변수를 평가하는 방법이다. =선택항목을 제외하고는 나머지선택항목들은 어떤 방법으로도 변수값을 변경시키지 못한다. 실례로 bname=${filename##*/}이나 fname=${flname:-emp.is}을 들수 있다.

19.10 쉘함수

쉘함수는 명령문들의 묶음으로 이루어 지는데 이 기능은 Bourne쉘에서도 유효하다. 이 함수는 지령렬을 단축하기 위한 쉘별명(17.4)보다 더 개선된것이다. 또한 별명으로서는 할수 없는 값을 되돌리는 기능도 가지고 있다.

```
함수이름(){
명령문들
return 값
}
```

함수정의는 ()의 앞에 놓이며 함수본체는 대괄호 {}에 의하여 닫겨 진다. 함수가 호출되면 본체안의 모든 명령문들이 실행된다. return명령문이 본체내에 있다면 함수의 성공이나 실패를 표현하는 값(문자

렬값은 아니다.)을 되돌려 준다. 쉘명령문들은 해석방식으로 실행되므로 쉘함수는 그것을 호출하는 명령문들보다 앞에 있어야 한다.

먼저 간단한 응용프로그람에 대하여 보기로 하자. 등록부안에 있는 대단히 많은 파일들을 보자면 흔히 ls -1 | more를 사용한다. 가끔 ls지령을 선택적인 파일이름들과 함께 쓸수도 있다. 이 지령렬은 쉘함수의 리상적인 후보로서 여기서는 그것을 ll이라고 명명한다.

```
$ ll () {
> ls -l $* | more
> }
```

정의에서는 ()를 요구한다 하여도 함수호출시에는 그것이 사용되지 말아야 한다. 인수와 함께 혹은 인수없이 함수를 호출할수 있다.

```
ll                    ls -l | more을 실행시킨다
ll ux3rd??            ls -l ux3rd?? | more을 실행시킨다
```

여기서 C쉘별명은 원만하게 동작하였을것이다. 그러나 별명은 자기의 한계를 가진다. 쉘스크립트와 비슷하게 쉘함수들은 지령행인수들을 사용할수도 있다($1, $2, 기타 등등과 같이). $*와 $#는 함수들에서도 자기의 의미를 가지고 있다. 그러나 별명은 이것들을 인식하지 못한다.

어디서 쉘함수를 정의하겠는가? 함수는 여러 곳에서 정의될수 있다.

- 그것을 사용하는 모든 스크립트의 시작위치에서 정의될수 있다.
- 현재대화에서 유효하도록 .profile에(혹은 쉘을 지적하는 시동파일에) 정의될수 있다.
- 다른 프로그람들이 그것을 사용할수 있도록 개별적인 《서고》파일에 정의될수 있다.

여기서 우리는 수속과 아주 비슷한 ll함수를 사용하였다. 그러나 쉘함수는 return문으로써 값을 돌려줄수 있다. 함수의 성공 혹은 실패를 의미하는 이 값은 $?에 보존된다.

주해

지령행인수들로부터 쉘스크립트에 의하여 설정되는 위치파라메터들은 쉘함수들에서는 직접 쓸수 없다. 그것들은 개별적인 변수들에 기억되거나 함수의 인수로서 넘겨 져야 한다.

```
ll $2
```

여기서 쉘스크립트의 두번째 인수($2)는 함수 ll의 첫번째 인수로서 넘겨 진다. 즉 $1은 함수내부와 외부에서 다른 의미를 표현한다.

19.11 쉘함수작성

지금까지는 몇개의 쉘프로그람들을 작성하면서 반복적인 지령렬을 리용하였다. 어떤 지령렬은 사용자가 처리를 계속하겠는가를 질문하거나 혹은 사용자입력의 유효성을 검사하였다. 이제 보게 될 부분들에서는 스크립트들에서 사용하려고 하는 유용한 몇가지 쉘함수들을 작성한다. 여기서는 쉘함수들을 쉽게 호출할수 있도록 이러한 함수들에 대한 《서고》를 작성할것이다.

19.11.1 체계날자로부터 파일이름의 생성

체계관리자라면 지적된 동작을 해야 할 매 날자들에 대하여 개별적인 파일들을 보존해야 한다. 이 파일이름들이 체계날자로부터 파생된다면 정해 진 날자에 해당한 파일을 쉽게 식별할수 있다. 파일이름

이 함수 dated_fname()에 의해 평가되도록 하는것은 좋은 착상이며 이 함수는 date출력값에 의해 파일이름을 차례로 이끌어 낸다. 그러면 이 함수를 프롬프트에서 정의하여 보자.

```
$ dated_fname () {
> set -- `date`
> year=`expr $6 : '..\(..\)'`          마지막 두개의 문자를 선택한다
> flname="$2$3_$year"
> }
```

여기서는 쉘함수들이 true 혹은 false의 값만을 돌려 주므로 변수 flname에 요구되는 값을 보관하였다. 더우기 함수내에 설정된 위치파라메터들은 현재쉘에서 유효하지 못하다. 함수를 실행시킨후에 flname의 값은 현재쉘에서도 쓸수 있어야 한다.

```
$ echo $flname
Sep22_00
```

이 문자렬을 매일 한개씩 유일한 파일이름을 작성하는데 사용할수 있다. Korn과 bash사용자들은 expr대신에 명령문 year=${6##??}을 리용하기 더 좋아할것이다. Oracle사용자들은 exp(export)지령에 대하여 체계가 발생시키는 덤프파일이름을 만드는데 이 함수를 사용할수 있다.

```
exp scott/tiger file=$flname
```

이것은 반출덤프파일 Sep22_00.dmp를 발생시킨다. 서고파일 mainfunc.sh를 만들고 거기에 함수정의를 놓으시오. 이 파일은 앞으로 여기에 두개정도의 함수들을 추가한후에 사용하기로 한다.

19.11.2 계속하기 위하여 또는 중지하기 위하여

값을 되돌리는 쉘함수에 대하여 보기로 하자. 그러면 스크립트 dentry1.sh의 마지막에 리용된 지령렬을 생각해 보시오. 이 스크립트는 사용자에게 계속하겠는가 혹은 제일 바깥순환을 완료하겠는가 하는 질문에 y 혹은 n을 누를것을 요구한다. 이 루틴은 스크립트내에서 자주 사용되므로 이것은 다음의 함수 anymore()로 바꾸는것이 더 좋다.

```
anymore () {
   echo "\n$1 ?(y/n) :\c" 1>&2              인수로 제공된 프롬프트
   read response
   case "$response" in
      y|Y) echo 1>&2 ; return 0 ;;
        *) return 1 ;;
  esac
}
```

함수는 어떤 질문인가 하는것을 결정하기 위하여 자기의 인수 $1을 리용한다. 이 함수가 wish to continue라는 문자렬을 인수로 가지고 호출되면 응답에 대한 질문은 다음과 같다.

```
$ anymore "Wish to continue"
```

```
Wish to continue ?(y/n) : n
$ echo $?
1                 return문에서 지적된 값
```

우리는 이 장의 뒤부분에서 이 함수의 돌림값을 사용하게 될것이다. 이 함수정의를 mainfunc.sh에도 놓으시오.

19.11.3 자료항목의 유효성검사

다시한번 스크립트 dentryl.sh를 보자(18.15.2). 자료항목이 유효한가를 반복적으로 질문하는 while 순환이 있다. 이 지령렬을 쉘함수로 변화시켜야 한다는것은 분명하다.함수 valid_string()은 두가지 검사를 진행해야 하는데 첫번째는 어떤것이 기입되였는가 하는것이고 두번째는 정확한 길이를 넘지 않았는가 하는 것이다.

```
valid_string () {
    while echo "$1 \c" 1>&2 ; do
         read name
         case $name in
           "") echo "Nothing entered" 1>&2 ; continue ;;
           * ) if [ `expr "$name" : '.'` -gt $2 ] ; then
                   echo "Maximum $2 characters permitted" 1>&2
               else
                   break
               fi ;;
         esac
    done
    echo $name
}
```

함수는 두개의 인수를 가진다. 첫번째는 프롬프트문자렬이고 두번째는 응답문자렬의 최대길이값이다. Korn과 bash사용자들은 expr지령을 사용하는 부분을 [${#name} -gt $2]로 바꾸려고 할것이다. 여기서는 이 함수를 서고파일 mainfunc.sh에 배치한다. 현재 이 서고파일에는 세개의 함수가 있는데 사용자는 점지령(.)으로 서고파일을 실행시켜 임의의 쉘스크립트안에서 이 함수들을 사용할수 있게 할수 있다. 그러면 스크립트 user_passwd.sh에서 우와 같은 기능을 리용해 보자.

```
$ cat user_passwd.sh
#사용자의 입력을 확인하는 스크립트.쉘함수를 두번 리용한다
. mainfunc.sh
user=`valid_string "Enter your user-id :" 16`
stty -echo
```

```
password=`valid_string "Enter your password:" 9`
tty echo
echo "\nYour user-id is $user and your password is $password"
```

이 자그마한 스크립트는 사용자의 이름과 통파암호를 접수하여 그것들이 길이로서 16과 9를 넘지 않는가를 확인하는것으로써 그것들의 유효성을 검사한다. 이 실례대화는 쉘함수들이 어떻게 스크립트의 크기를 줄이는가 하는것을 보여 준다.

```
$ user_passwd.sh
Enter your user-id : robert louis stevenson
Maximum 16 characters permitted
Enter your user-id : scott
Enter your password:
Nothing enthered
Enter your password: *****                화면에 보이지 않는다
Your user-id is scott and your password is tiger
```

쉘함수들은 위치파라메터들을 비롯하여 모든 쉘구조들을 받아 들이므로 쉘스크립트들을 복귀시킬수 있다. 쉘함수는 기억기에 상주되므로 이것에 대한 호출은 디스크의 입출력회수를 줄인다. 더우기 쉘함수는 현재쉘에서 실행되므로 쉘에 정의된 변수들(위치파라메터는 아니다.)은 스크립트에서는 물론 함수내에서도 볼수 있다. 그러나 현재의 가입대화에 대하여 자기가 요구하는것만큼 많은 함수들을 호출할수 있는가를 확인해 보시오.

참고

많은 쉘함수들이 여러개의 프로그람들에서 쓰일 때에는 그것들모두를 단일한 《서고》파일에 배치하여야 하며 편리한 위치에 파일을 기억시켜야 한다. 이 함수들을 요구하는 모든 스크립트의 시작부분에 점지령으로써 서고파일을 실행시키는 명령문을 삽입하시오.

19.12 지령행을 두번 평가하기(eval)

변수를 관흐름으로 설정하고 그것을 실행시켜 본적이 있는가? 다음과 같이 해보자.

```
cmd="ls | more"
$ cmd                  |와 more는 ls에 대한 인수이다
```

이것은 우리가 바라던 페지화된 출력을 하지 않는다. 그러면 《수값화된 프롬프트》를 정의하고 그것을 평가해 보자.

```
$ prompt1="User Name:" ; x=1
$ echo $prompt$x                $프롬프트는 정의되지 않는다
```

첫번째 경우에는 쉘이 |기호와 more를 ls의 인수로 식별하고 그후에 변수를 평가하였다. 결과적으로 ls는 그것들을 두개의 인수로서 취급하여 예측할수 없는 출력을 산출한다. 두번째 실례에서 쉘은 먼저 정의되지 않은 $프롬프트를 평가하며 다음값이 1인 $x를 평가한다.

이 지령렬들을 정확히 실행시키자면 지령행에서의 값평가를 뒤로 미루어야 한다. 이를 위하여 우리는 지령행을 두번 평가하는 eval명령을 리용해야 하며 이것은 Bourne쉘에서도 유효하다. 즉 첫 경로에서 값평가를 하지 않고 두번째 경로에서만 그것을 평가한다.

첫번째 실례의 지령렬을 eval명령문을 리용하여 다음과 같이 실행시킬수 있다.

```
eval $cmd
```

첫번째 경로에서 eval명령문은 세개의 인수들인 ls, |, more들을 찾는다. 다음 그 지령행을 재평가하며 |주위에 있는것들을 두개의 지령으로 분할한다. 지령은 정확히 실행된다.

두번째 경우에는 \로써 첫번째 $를 은폐시키고 다음 eval을 리용하여 동작하게 할수 있다.

```
$ x=1 ; eval echo\$prompt$x
User Name:
```

첫번째 경로는 \에 의하여 의미해제된 $를 무시하며 결과적으로 평가는 \$prompt1으로 된다. 두번째 경로에서는 \을 무시하며 우리가 바라는대로 변수 $prompt1을 평가한다. 여기로부터 우리는 여러개의 수가 붙은 변수들을 리용하는 응용프로그람을 생각해 볼수 있다. 스크립트가 말단으로부터 입력자료를 10번 받아 들여야 한다면 자료를 기억시키기 위하여 10개의 변수들을 정의하고 사용해야 한다. 때때로 실행시에 요구되는 변수의 수가 임의로 될수 있다.

사람들은 변수이름 그자체가 스크립트내에서 자동적으로 발생되게 하는 더 일반적인 스크립트를 가질것을 요구한다. 그러면 우리는 질문들을 변수들처럼 기억한 다음 《수값화된 변수들》에로 자료를 읽어들여야 할것이다.

아래에서는 수값화된 프롬프트를 리용하였다. 즉 여기서는 valuel, value2, value3 등과 같은 수값화된 변수들을 사용할 필요가 있다.

```
$ { x=1
>eval echo \$prompt$x '\\c'
>real value$x
>eval echo \$value$x ; }
User Name: kleinrock
kleinrock
```

명령문 read value $x는 변수 valuel에로 응답을 읽어 들이며 eval명령문은 두번째 경로에서 이 값을 현시한다. 이것은 다음의 스크립트에서 가치가 있는 중요한 결과로 된다.

$1 혹은 임의것으로써 위치파라메터들을 호출할수 있다. 그러면 마지막파라메터는 직접 호출할수 있겠는가? $#의 값을 가지고 있기때문에 eval에 대한 봉사를 리용할수 있다.

```
$ tail -l /etc/passwd
martha:x:605:100:martha mitchell:/home/martha:/bin/ksh
$ IFS=:
$ set `tail -l /etc/passwd`                여기서는 set가 필요없다
$ eval echo \$$#
```

```
/bin/ksh
```

우에서 보는바와 같이 우리는 파일 /etc/passwd에서 한개 행에 대한 마당의 수를 알지 못한다. 우리는 이와 같은 방법으로 마지막스크립트인수를 쉽게 분리시킬수 있다. 그러면 목록으로부터 등록부를 제거하기 위한 스크립트 cpback2.sh에(18.17) head와 tail명령문을 사용할 필요가 있겠는가?

이 장에는 이러한 화제에 대한 련습이 있다.

19.13 사용자등록자리를 만드는데서 eval의 리용

useradd지령(22.3.2)으로 사용자등록자리를 만드는 스크립트를 개발해 보자. 이것은 뿌리허가권을 요구하는 이 장에서 유일한 스크립트(19.2에서 론의된 fdisk는 뿌리에서도 실행된다.)로 된다. useradd에 의하여 필요되는 모든 인수들은 대화식으로 제공된다. 이 스크립트(createuser)는 다음의 특징을 가진다.

- 수값화된 프롬프트와 변수를 리용하여 6개의 마당으로 입력자료를 받는다.
- 파일 mainfunc.sh에 있는 함수 anymore()를 리용한다.
- 쉘이 bash인가 그렇지 않은가에 따라 echo명령문이 -e선택항목과 함께 리용되는가 하는것을 결정한다.

사용자에게 여섯번 질문해야 하기때문에 정밀한 스크립트를 만들자면 eval을 사용하는것이 좋다. creatuser.sh스크립트는 그림 19-3에 보여 주었다. 여기서 mainfunc.sh는 좀 다르게 실행된다. 현재등록부는 일반적으로 상급사용자의 변수 PATH에 설정되여 있지 않다. bash를 제외한 모든 쉘에서 점지령은 현재등록부의 스크립트에 대한 탐색을 하지 못한다. 그러므로 상대경로를 리용해야 하였다.(17.9의 주해)

여섯개의 프롬프트는 수값화된 변수들로서 스크립트의 시작위치에 정의되여 있다. 안쪽의 while순환은 그것들모두를 차례로 출력한다. 값은 변수 valuel, value2 등으로 읽혀 진다. 그러면 사용자 ppp를 만들어 보자.

```
# ./createuser.sh
User Name: ppp
Group-id: 100
Home Directory: /home/ppp
Login Shell: /etc/ppp/ppplogin
GCOS Details: PPP Server Account
More users to create ?(y/n) : n
```

여기서 새로운 사용자등록자리가 만들어 졌지만 그래도 /etc/passwd의 마지막행을 보는것으로써 확인하자.

```
# tail -l /etc/passwd
ppp:x:520:100:PPP Server Account:/home/ppp:/etc/ppp/login
```

```
# 사용자를 만드는 프로그람—프롬프트들과 변수들을 만드는 eval을 리용하였다
# 변경하지 않고 Korn, bash쉘들에서 실행시킬수 있다
```

```
option=
[ $SHLL = "/bin/bash" ] && option=-e
. ./mainfunc.sh                                    # anymore() 함수를 유효하게 한다

prompt1="User Name:"     ; prompt2="User-id:"    ; prompt3="Group-id:"
pormpt4="Home Directory:" ; prompt5="Login Shell:" ; prompt6="GCOS Details:"

while true ; do
   x=1
   while [ $x -le 6 ] ; do
      eval echo $option \$prompt$x '\\c' 1>&2           # 6개의 프롬프트를 만든다
      read value$x
      x=`expr $x + 1`
   done
   useradd -u $value2 -g $value3 -d $value4 -s $value5 -c "$value6" -m $value1
   anymore "More users to create" 1>&2 || break
done
```

그림 19-3. eval을 리용한 사용자등록자리만들기스크립트(createuser.sh)

우리는 최소규모의 스크립트를 가지고 여섯번의 사용자응답을 여섯개의 변수들에 읽어 들였다. echo는 bash가 사용자의 가입쉘이라면 자동적으로 -e선택항목을 사용하므로 이 스크립트는 모든 쉘들에 대하여 변경하지 않아도 된다. 현재등록부를 상급사용자의 PATH에서 찾을수 없으므로 스크립트를 ./createuser.sh로 실행시켜야 하였다.

주해

ppp사용자를 만들 때 정의되는 가입쉘은 쉘이 아니다. 정확히 말한다면 실행가능한 어떤 지령일수 있으며 쉘에 속박되지 않아도 된다. 만일 독자가 PPP봉사기를 만들려고 한다면 쉘에 대한 사용자접근을 허용하지 말아야 한다. 가입프로쎄스는 ppplogin스크립트를 실행시키며 그때 사용자를 탈퇴시킨다. 이것은 일반적으로 해야 하는 방법이다.

19.14 exec명령문

프로쎄스만들기기구에 대한 학습에서(10.2) 생성된 프로쎄스를 덧놓는 exec체계호출에 대하여 보았다. 이것은 때때로 다른 프로그람코드로 현재쉘 그자체를 덧쓰기해야 하는 쉘스크립트작성자들에게 있어서 중요하다. 지금까지는 할수 없었던것이지만 임의의 UNIX지령앞에 exec를 쓰면 그 지령은 현재쉘을 덧쓰기한다. 이렇게 하면 지령이 완료될 때 체계에서의 탈퇴를 일으키게 된다.

```
$ exec date
Sun Apr 9 14:12:03 EST 2000
login:
```

때때로 우리는 사용자가 가입할 때 단일프로그람을 자동적으로 실행시키며 쉘에로 탈퇴되지 않게 하고 싶을 때가 있다. 그 지령을 파일 .profile에, exec뒤에 배치할수 있다. 사용자가 체계에 가입하면 지

령이 기동되고 그 지령실행이 완료되면 그는 체계에서 탈퇴하게 된다. 앞의 실례에서는 첫 위치에 쉘이 없었다.

exec는 현재쉘을 다른것으로 바꾸는데 쓸모가 있다. exec ksh를 사용하면 현재의 쉘환경은 완전히 새로운 쉘로 교체된다. 또한 exec login userid로써 다른 사용자등록자리에 가입할수도 있다. 이 기술은 telnet를 가지고 원격기계우에서 작업할 때 효과적이다.

현재쉘에서의 방향절환효과

exec는 또 하나의 중요한 특징을 가진다. 그것은 전체 스크립트에 대하여 표준흐름들을 방향절환시킬수 있다. 만일 스크립트에 표준출력이 단일파일로 되여 있는 여러개의 지령들이 포함되여 있다면 그 매개에 대하여 방향절환기호를 리용하는것이 아니라 exec를 리용하여 아래와 같은 방법으로 기정목적지를 재할당할수 있다.

```
exec > found.lst                    >>도 리용할수 있다
```

여기서 우리는 스크립트 그자체의 방향을 절환하였다. 그러나 exec는 표준적으로 제공되는 파일서술자를 가진 3개의 흐름(0, 1, 2)을 제외한 개별적인 흐름들을 만들수 있다. 실례로 모든 출력을 지적하기 위한 파일서술자 3을 만들수 있으며 그것을 물리적파일 foundfile과 결합시킬수 있다.

```
exec 3>foundfile
```

이때 표준출력흐름을 파일서술자 3과 결합시켜 파일에 써넣을수 있다.

```
echo "This goes to foundfile" 1>&3
```

프로그람작성언어에서 이것은 exec를 파일에 대한 론리적이름을 제공하는데 사용할수 있다는것을 의미한다(perl에서는 filehandle). 그러므로 이 강력한 입출력처리기를 가지고 파일들을 더 간결하고 세련된 방법으로 처리할수 있을것이다.

그러면 파일에서 empID(종업원ID)를 읽어 내는 스크립트를 설계하여 보자. 즉 emp.lst를 탐색하여 다음의것들을 각각 세개의 파일들에 보관하자.

- 탐색된 행들
- 탐색하지 못한 empID들
- 틀린 형식의 empID들

우선 다음의 empID들을 포함하는 파일을 보기로 하자. 이 파일에는 스크립트에 의하여 제거되여야 하는 두개의 세자리수로 된 empID들이 들어 있다.

```
$ cat empid.lst
2233
9765
2476
789
1265
9877
```

```
5678
245
2954
```

그림 19-4에 제시된 countpat.sh스크립트는 표준출력을 3개의 흐름으로 나누고 그것들을 3개의 개별적인 파일에로 방향절환한다. 여기에는 4개의 인수들이 요구되는데 하나는 패턴을 포함하는 파일이고 나머지 3개는 흐름에 대한 파일들이다.

```
exec > $2                         #선택된 행들을 보여 주기 위하여 파일 1을 연다
exec 3> $3                 #탐색하지 못한 패턴들을 보여 주기 위하여 파일 3을 연다
exec 4> $4                      #무효한 패턴들을 보여 주기 위하여 파일 4를 연다

[$# -ne 4 ] && { echo "4 arguments required" ; exit 2 ; }

exec < $1                                                      # 입력방향절환
while read pattern ; do                            # read는 $1로부터 읽어 들인다
    case "$pattern" in
    ????) grep $pattern emp.lst ||
          echo $pattern not found in file 1>&3 ;;
       *) echo $pattern not a four-character string 1>&4 ;;
    esac
done
exec 0<&- ; exec >&- ; exec 3>&- ; exec 4>&-                     # 파일을 닫는다
exec >/dev/tty                               # 표준출력방향을 다시 말단에로 돌린다
echo Job Over
```

그림 19-4. exec를 리용하여 다중흐름을 만드는 스크립트(countpat.sh)

표준출력흐름은 파일서술자 1, 3, 4와 결합된다. 또한 $1을 모든 표준입력의 원천으로 설정하였다. 이것은 순환안에서의 read명령문이 $1(패턴을 포함하는 파일)로부터 입력을 가진다는것을 의미한다. 일단 모든 파일의 쓰기가 완료되면 표준출력흐름은 말단(exec >/dev/tty)에로 다시 할당되여야 한다. 그렇지 않으면 Job Over라는 통보문이 $2로 지적된 파일에 보존될것이다.

이 스크립트는 아주 명백하며 결합기호를 리용한 두개의 명령문을 가지고 있다. grep명령문은 표준출력파일서술자를 리용하므로 결합이 필요 없다. 스크립트는 4개의 인수를 가지고 3개의 파일로 출력자료를 분할한다.

```
$ countpat.sh empid.lst foundfile notfoundfile invalidfile
Job over
```

통보문은 3개의 파일중 어느하나에 기록되는것이 아니라 말단에 표시된다. 그러면 이 3개의 파일을 찾아서 실제로 무엇이 기록되여 있는가를 보자.

```
$ cat foundfile
```

```
2233|charles harris   |g.m.      |sales     |12/12/5 | 90000
2476|jackie wodehouse |manager   |sales     |05/01/59|110000
1265|p.j. woodhouse   |manager   |sales     |09/12/63|  90000
5678|robert dylan     |d.g.m     |marketing |04/19/43| 85000
$ cat notfoundfile
9765 not found in file
9877 not found in file
2954 not found in file
$ cat invalidfile
789 not a four-character string
245 not a four-character string
```

이것이 바로 exec명령문이 가지고 있는 능력이다. 스크립트는 여러개의 파일을 동시에 열어서 perl이 자기의 파일처리자들을 리용하는것과 같은 방법으로 개별적으로 매개 파일들을 호출하였다. 스크립트에서 파일이름이 아니라 파일서술자를 리용하는것이 더 좋다. 그것은 쉘스크립트들이 파일들에 대하여 독립성을 부여하기때문이다.

19.15 쉘스크립트에 대한 오유수정(set -x)

쉘은 스크립트의 오유수정(debug)에 사용되는 기능인 -x선택항목을 제공한다. 그것이 스크립트내에서(지어는 $프롬프트에서) 리용되면 말단에 매개 명령문을 출력하는데 실행될 때는 +기호가 명령문앞에 붙는다. 그러면 cpback2.sh스크립트(18.17)를 그의 시작부분에 아래와 같은 명령문을 놓는것으로써 수정하자.

```
set -x
```

set +x는 set -x를 해제하며 스크립트의 마지막위치에 놓을수 있다. 그러면 오직 하나의 파일이름 index와 함께 스크립트를 실행시켜 보자(파일 index는 등록부 safe에 이미 존재하므로 index.2에 복사된다). 설명문이 붙은 출력결과를 아래에 보여 준다.

```
$ cpback2.sh index safe
+ [ 2 -lt 2 ]                인수의 수를 검사한다
+ tr ' ' '\012'
+ echo index safe                이것은 echo $* 이다
+ 1>707
+ tail -l 707                등록부이름을 식별하고 그것을
+ destination=safe                변수에 설정한다
+ [ ! -d safe ]
+ expr 2 - l
+ count=1
```

```
+ head -l 707              하나의 파일만 복사
+ [ ! -f safe/index ]
+ copies=1
+ true
+ [ ! -f safe/index.1 ]
+ expr 1 + 1
+ copies=2              index.1은 존재한다
+ true
+ [ ! -f safe/index.2 ]
+ expr 2 + 1
+ copies=3              index.2 역시 존재한다
+ true
+ [ ! ?f safe/index.3 ]
+ cp index safe/index.3
+ echo 'File index copied to index.3
+ break
+ rm 707              림시파일이 제거된다
```

이것은 스크립트가 왜 정확히 동작하지 않는가 하는것을 알아 내기 위한 리상적인 도구이다. 쉘은 매개 명령문이 실행되면 앞에 +기호를 첨부하여 화면에 출력한다. 즉 head, tail, cp지령들에 대한 지령행을 보여 주며 지어는 expr지령이 순환이 반복될 때마다 변수 copies의 값을 어떻게 증가시키는가 하는것을 보여 준다.

19.16 프로그람에 대한 새치기(trap)

쉘스크립트들은 기정적으로 새치기건이 눌리우면 처리를 완료한다. 이런 방법으로 쉘스크립트들을 완료하면 디스크에 림시파일들이 많이 남으므로 좋은 방법이 못된다. trap명령문을 리용하여 스크립트가 어떤 신호를 받는 경우에만 요구하는 처리를 진행하게 할수 있다. 보통 이 명령문은 쉘스크립트의 시작부분에 놓이며 2개의 목록을 리용한다.

```
trap 'command_list' signal_list
```

스크립트가 signal_list의 신호들중 임의의 신호를 받으면 trap명령문은 command_list의 지령들을 실행한다. 신호목록은 kill지령에서 리용하였던것과 같은 하나이상의 신호들에 대한 옹근수값 혹은 이름을 포함할수 있다. 이로부터 신호목록을 2 15 혹은 INT TERM이라고도 쓸수 있다.

만일 습관적으로 쉘의 PID번호를 리용한 이름을 가지고 림시파일을 만든다면 새치기가 발생할 때마다 그것들을 제거하기 위하여 trap명령문의 봉사를 사용해야 할것이다.

```
trap 'rm $$* ; echo "Programe interrupted" ; exit' 1 2 15
```

그러면 신호 1, 2 혹은 15를 보낼 때마다 trap는 신호를 가로 채서(포착하여) $$*에 맞는 모든 파일들을 제거하고 통보문을 현시한 다음 스크립트를 완료한다. 새치기건이 눌리워 지면 신호 2가 발생된다. 모든 스크립트들에 이 번호를 포함시키는것은 아주 좋은것으로 된다.

또한 신호를 무시하고 처리를 계속하도록 요구할수도 있다. 이 경우에는 빈 지령목록을 사용하여 이와 같은 신호들이 영향을 받지 않게 할수 있다.

```
trap ' ' 1 2 15                    스크립트는 제거되지 않는다
```

trap명령을 쉘스크립트들에 반드시 리용해야 한다는것은 아니다. 그러나 그것을 가지고 있는 지령목록의 끝에 exit명령을 포함시키는것을 잊어서는 안된다. Korn과 Bourne쉘들은 등록에서 탈퇴할 때 파일을 실행시키지 않지만 trap를 리용하면 파일을 실행시킬수 있다. 이때에는 신호번호 0을 리용해야 한다. 이러한 쉘들에서는 신호들을 기정값들로 재설정하는데 trap -명령문을 사용할수 있다. 또한 한 스크립트에서 여러개의 trap지령들을 사용할수 있는데 그 매개는 이전의것을 무시한다.

쉘에 대한 학습은 이것으로 끝낸다. UNIX체계의 능력이 충분히 발휘되자면 awk, perl과 함께 쉘역시 심중히 취급되여야 한다. 쉘프로그람들이 C프로그람들보다 속도가 뜨다고는 하여도 관리측면에서의 과제들에서는 대체로 속도문제가 중요한것으로 되지 않는다.

요 약

set는 위치파라메터들에 값을 설정하며 shift는 그것들을 왼쪽으로 밀기한다. 제일 왼쪽의 변수는 류실되며 shift를 사용하기전에 어떤 변수에 보존하여야 한다. set에서 사용되는 마당경계구분문자는 IFS변수에 의하여 결정된다. set --은 출력이 null이거나 기호 -으로 시작되는 경우 기정적인 기능 혹은 선택항목으로 해석하는것을 막기 위하여 사용된다.

here문서(<<)는 스크립트 그자체에서 스크립트에 대한 입력을 제공한다. 또한 지령대입 혹은 변수들과 함께 사용될수 있다. 흔히 파일이름을 인수로 가지고 있지 않는 지령과 함께 혹은 대화형프로그람들을 비대화형으로 실행시키기 위하여 리용한다. let는 Korn쉘과 bash의 내장기능으로서 계산능력을 가지고 있다. 값을 대입할 때 앞붙이 $는 필요 없다. Korn쉘과 bash는 추가적인 POSIX호환계산도구로서 ((와))연산자들을 사용한다.

조건문 혹은 순환문은 fi나 done열쇠단어에서 방향절환되거나 관련결될수 있다. 절환구조의 내부에서 말단을 가리키는 표준출력은 >/dev/ttty에 의하여 개별적으로 절환되여야 한다. 쉘은 또한 기호렬 1>&2와 2>&1을 사용하여 표준출력과 표준오유흐름을 결합할수 있다.

어미프로쎄스에서 정의된 변수는 그것이 반출될 때에만 새끼프로쎄스에서 볼수 있다. 그러나 새끼프로쎄스에서 그 변수의 값을 변경시키면 그 변화된 값은 어미프로쎄스에서는 볼수 없다. 정합연산자들인 ()는 보조쉘에서 지령묶음을 실행시키지만 {}는 보조쉘을 기동시키지 않는다.

Korn과 bash는 1차원배렬을 제공한다. 배렬은 날자의 유효성검사에 리용할수 있다.

Korn과 bash에서는 문자렬처리기능이 훌륭히 개발되여 있다. 변수내용은 통용기호로서 정합시킬수

있다. 문자 #과 %를 리용하면 문자렬의 시작과 끝에서 패턴을 정합할수 있으며 비정합부분을 추출할수 있다.

쉘변수들은 0 아닌 값을 가지고 있는가 그렇지 않은가 하는데 따라 조건방식으로 평가될수 있다. ? 연산자는 오유통보문을 현시하고 쉘에로 탈퇴하며 =연산자는 변수에 값을 할당한다.

쉘의 함수를 리용하면 중요하고 반복적인 지령렬들을 함축할수 있으며 이것은 별명보다 더 강력하다. 또한 함수는 위치파라메터를 받아 들이지만 인수로서 그것을 넘기지 않는 한 스크립트에까지 넘겨 지지 않는다. 함수는 true나 false값만을 돌려 준다.

eval명령문은 지령행을 두번 반복하여 처리하며 배렬을 모의하거나 변수들을 실행시키는데 리용된다. eval로 일반화되고 수값화된 프롬프트와 변수들을 만들수 있다.

exec가 지령의 앞에 놓이면 현재의 쉘을 겹쳐놓기한다. 또한 여러개의 흐름을 만들수 있으며 그것들을 파일서술자와 결합시킬수 있다. 실례로 1>&7을 써서 비표준파일서술자 7에로 쓰기를 진행한다. 쉘스크립트들에 대하여 오유수정을 하자면 모든 지령행이 화면에 표시되도록 스크립트의 시작위치에 set -x를 놓아야 한다. 이 지령은 순환의 전체 반복과정을 보여 준다.

trap를 리용하여 스크립트가 특수한 방법으로 새치기에 응답하도록 할수 있다. 이것은 스크립트가 신호를 받는 경우 림시파일들을 제거하는데 유용하다. 또한 자기의 스크립트가 새치기의 영향을 받지 않게 할수 있다.

시험문제

1. 아래의 지령은 무엇을 수행하는가?

    ```
    set `set`
    ```

2. 스크립트가 12개의 파라메터를 리용한다면 마지막파라메터를 어떻게 호출할수 있겠는가?
3. 다음지령에서 문제로 되는것은 무엇인가?

    ```
    set `grep -c "A HREF" catalog.html`
    ```

4. 쉘스크립트내에서 등록부를 변경시킨 경우 왜 스크립트실행이 완료되면 본래의 등록부가 다시 회복되는가? 이러한 문제를 극복하자면 어떻게 해야 하는가?
5. 지령프롬프트에 변수를 정의하는 경우 그의 값을 쉘스크립트내에서 어떻게 유효하게 할수 있겠는가?
6. script지령을 호출하고 프롬프트에 변수를 정의한 다음 exit로 스크립트에서 탈퇴하시오. 다음 변수의 값을 출력하시오. 거기서 무엇을 볼수 있으며 왜 그렇게 되는가?
7. 지령 echo ${#x}은 x:Undefined variable이라는 통보문을 출력한다. 어느때 이렇게 되며 이 표현식은 무엇을 위해서 설계되였는가
8. 다음의 명령문은 무엇을 하는가?

    ```
    flname=${l:-emp.lst}
    ```

9. 어느때 지령에 here문서가 필요한가?

10. 쉘함수를 호출하여 현재등록부를 제거하기 위한 스크립트를 작성하시오.

11. 스크립트 var.sh(19.6.1)에서 export를 쓰지 않고 어떻게 변수 x가 외부에서 설정한것과 같은 값을 가지게 할수 있는가?

12. 체계에 가입하자마자 특정의 프로그람이 실행되고 프로그람이 완료될 때 사용자가 체계에서 탈퇴된다는것을 어떻게 확인할수 있겠는가?

련습문제

1. 지령 set `cat foo`이 unknown option이라는 오유통보문을 발생시켰다면 파일 foo가 읽기가능한 파일이라고 생각할수 있는 리유는 무엇인가?

2. 인수로서 파일이름을 받아 그 파일이 존재하면 마지막변경날자를 표시하며 그렇지 않으면 적당한 통보문을 현시하는 스크립트를 작성하시오.

3. 한개 또는 여러개의 단어패턴을 인수로 받아 파일들의 묶음(foo*)에서 그것을 탐색하고 그 패턴을 포함하는 파일들과 함께 vi편집기를 기동시키는 스크립트를 작성하시오. 매 파일에서 패턴을 어떻게 찾겠는가?

4. 닉명ftp싸이트(ftp.planets.com과 같은)와 임의의 개수의 ftp지령("cd pub", "get cp32.tar. g2"와 같은)들을 인수들로 받아 들이는 스크립트를 작성하시오. 다음 이것을 인터네트싸이트에 련결하고 자동적으로 가입한 다음 ftp지령들을 실행시키시오.

5. 현재등록부에서 재귀적으로 패턴을 탐색하도록 3번 문제의 스크립트를 다시 작성해 보시오.

6. 아래와 같이 쉘스크립트에 exit지령이 놓일 때 이 지령은 왜 스크립트를 완료시키지 못하는가? 이를 위해서는 어떻게 해야 하겠는가?

```
( statements; exit )
```

7. su지령을(뿌리통과암호를 알고 있는 경우) 호출한 다음 말단이름과 함께 ps -t를 실행시키시오. 어떤 충돌이 있겠는가?

8. 다음의 두행을 포함하는 작은 스크립트 cman이 있다고 하자.

```
#!/bin/ksh
x=`find $HOME -name $1 -print`
cd $x
```

cman manl을 실행시키면 등록부 manl이 홈등록부내부안의 어딘가에 있는데도 불구하고 현재등록부가 변경되지 않는다. 왜 이런 현상이 생겼으며 어떻게 등록부를 변경시킬수 있는가? Bourne쉘에서는 어떻게 해야 하는가?

9. 다음의 명령문에서 틀린것은 무엇인가? 정확하게 실행시키자면 어떻게 변경시켜야 하는가?

```
[ $# -ne 2 ] && echo "Usage: $0 min_guid max_guid" ; exit
```

10. 야간에 일감을 실행시켜야 하는 경우 출력과 오유통보문을 둘 다 같은 파일에 보관시키 위해서는 스크립트를 어떻게 실행시켜야 하겠는가?
11. exec를 리용하여 한 파일에는 스크립트의 출력을, 다른 파일에는 오유통보문을 보관하는 스크립트를 어떻게 작성할수 있겠는가?
12. 배렬을 사용하여 스크립트에 대한 마지막지령행인수를 어떻게 추출할수 있는가?
13. 명령문 while [${count:=1} -lt 50]을 가지고 있는 스크립트는 순환을 전혀 실행시킬수 없다. 원인은 무엇인가?
14. 세개이상의 파일이름들과 함께 rm지령을 리용할 때마다 대화식으로 동작하도록 하는 쉘함수를 작성하시오.
15. 인수들에 지적된 파일들(인수가 없으면 모든 파일)의 전체 크기를 보여 주는 쉘함수 lstot()를 작성하시오.
16. 19.13의 실례를 eval명령문대신에 배렬을 사용하여 작성해 보시오.
17. 호출한 프로그람에로 조종을 넘기기 위하여 함수내부에서 왜 exit명령문을 리용할수 없는가?
18. 스크립트 cpback.sh(18.17)를 파일이 등록부에 없는 경우에만 거기에 복사하도록 수정하시오. 오직 하나의 외부지령 cp를 리용하며 Korn과 bash쉘들의 기능들을 개발할수 있다. 스크립트에 대한 마지막인수는 등록부이다(참고: 등록부이름을 식별하는데는 eval명령을 사용하시오).
19. 18번 문제에서 개발된 프로그람을 오직 낡은 파일들만을 덧쓰기하도록 수정하시오.
20. 순환을 리용한 스크립트에 대하여 변수값의 변화과정을 보려고 하는 경우 echo명령문을 사용하지 않고 어떻게 볼수 있겠는가?
21. 새치기로 탈퇴하기전에 스크립트가 사용자에게 질문하기 위해서는 어떻게 해야 하는가?
22. Bourne 혹은 Korn쉘들에서 수값으로 시작되는 이름을 가진 모든 파일들이 탈퇴시에 다 제거되였다는것을 어떻게 확인할수 있겠는가?

제 20 장. 기본조작기 perl

perl은 UNIX에 가장 마지막으로 추가된것으로서 기능이 강한 도구들중의 하나이다. perl은 래리 월(Larry Wall)에 의하여 개발되였으며 각이한 용도에 쓰이는것으로 해서 UNIX체계에서 흔히 《스위스장교의 칼》로 불리우고 있다. 그 각이한 용도란 실용추출 및 보고서작성언어(Practical Extraction and Report Language)로 확장되였다는것을 의미한다. 그러나 이것은 원래의 목적을 초월한것으로 된다. 월은 perl에서 프로그람작성언어이기도 하고 모든 려파기들의 모체이기도 한 일반적인 도구를 창안하였다. perl은 모든 체계들에서 다 쓸수 있는것은 아니지만 Linux와 Solaris 8에서 표준장비되여 있다. 그리고 자유로우며 모든 UNIX변종들에 대하여 유효하다.

perl은 가장 강력한 UNIX도구들인 shell, grep, tr, sed, awk의 일부 기능들을 결합하였다. 사실상 이러한 도구들이 할수 없는것이면 perl도 할수 없다. 또한 파일, 등록부, 프로쎄스 등과 관련된 수백가지 기능도 가지고 있는데 그것들중 대부분은 UNIX와 C에서 한짝을 이룬다. perl은 또한 우리가 지금까지 론의한 모든 정규식들을 다 인식한다. 대단히 큰 실행파일에 대해서도 perl은 쉘과 awk보다 속도상 더 빠르다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 간단한 perl프로그람을 소개한다(20.1).
- chop를 리용하여 행 혹은 변수의 마지막문자를 제거한다(20.2).
- 확장문자렬과 련결연산자들의 사용방법을 배운다(20.3).
- 보다 우월한 문자렬처리기능에 대하여 배운다(20.4).
- 지령행에서 그리고 파일을 읽어 들이는 스크립트내에서 순환을 지정하는 방법을 배운다(20.5).
- 기정변수 $_의 의미를 배운다(20.6).
- 목록과 배렬 그리고 그의 연산자들의 리용방법을 배운다(20.8).
- 목록과 함께 동작하는 foreach순환에 대하여 배운다(20.10).
- split와 join에 의한 행의 분할과 결합방법을 배운다(20.11, 20.12).
- 비수값화된 첨자를 가진 결합적인 배렬의 처리방법을 배운다(20.14).
- 정규식 그리고 s와 tr지령들에 의한 치환처리방법을 배운다(20.15).
- 파일조종자를 리용한 파일 혹은 흐름접근방법을 배운다(20.16).
- 파일속성검사방법을 배운다(20.17).
- 반복사용을 위한 부분루틴의 개발방법을 배운다(20.18).
- perl기능을 리용하여 열람기로부터 기동되는 CGI프로그람을 개발하는 방법을 배운다(20.20, 20.21).

주해

perl에 대하여 리해하자면 많은 기능들이 이미 awk에 대한 장에서 설명되였거나 C교과서들에서 설명되여 있으므로 C 혹은 적어도 awk에 대한 지식이 있어야 한다. 이 장의 몇가지 실례들을 리해하자면 이전에 취급한 정규식에 대해서도 잘 알아야 한다. 필요하다면 시작에 앞서 15장과 16장의 내용을 다시 보는것도 좋다.

참고

vi편집기에서 perl스크립트들을 기동시킬수 있다. 이 기술은 18.2의 《참고》에 자세히 설명되였다.

20.1 perl의 기초

perl프로그람은 특수한 해석방식으로 실행된다. 즉 전체 스크립트는 실행되기전에 내부적으로 기억기에서 콤파일된다. 쉘, awk와 같은 다른 해석언어들과 달리 스크립트의 오유는 그의 실행전에 발생된다.

Linux에서 perl은 등록부 /usr/bin에 들어 있지만 다른 UNIX체계들에서는 다른 등록부 즉 될수록 /usr/cal/bin에 놓는다. 만일 변수 PATH에 그 등록부를 가지고 있다면 그것이 동작하는가 하는것을 검사하기 위한 방법은 다음과 같다.

```
$ perl -e 'print ("GNUs Not Unix\n");'
GNUs Not Unix
```

여기서 perl은 GNU략어를 현시하는데서 려과기처럼 기동한것이 아니라 echo와 아주 류사하게 동작하였다. awk와는 달리 출력은 perl의 기정동작이 아니므로 그것을 명백히 지적해 주어야 한다. C에서처럼 perl의 모든 명령문들은 반두점으로 끝난다.

perl에서 -e선택항목을 리용하면 지령행에서 아주 효과적인 처리를 많이 할수 있다. 대부분의 perl프로그람들은 크며(일반적으로 아주 크다.) .p1파일들로 배치된다. 아래에 변수들을 리용하여 연산을 진행하는 간단한 실례프로그람을 보여 준다.

```
$ cat sample.pl
#!/usr/bin/perl
print ("Enter your name: ");
$name = <STDIN>;                #건반으로부터 입력
print ( "Enter a temperature in Centigrade: ");
$centigrade=<STDIN>;                #공백은 중요치 않다
$fahrenheit=$centigrade*9/5 + 32;                #여기서도 마찬가지이다
print "The temperature $name in Fahrenheit is $fahrenheit\n";
```

첫행은 이 스크립트를 실행시키는데 리용되는 프로그람을 정의한다. 즉 쉘은 여기서 자기자체가 아니라 perl을 사용한다. 쉘스크립트들에서 이와 류사한 기능을 사용하였다.

그러므로 이것에 대하여 더 설명할 필요는 없다고 본다. 모든 perl프로그람의 첫행에 이 명령이 있는지 확인하시오.

perl변수들에 대해서는 그의 값평가(The temperature, $name...)에서는 물론 정의($name = <STDIN>)에서도 앞에 $이 붙어야 한다. <STDIN>은 표준출력을 표현하는 파일조종자(filehandle-파일의 론리적이름)이다.

우의 프로그람에서 마지막 print명령은 괄호를 사용하지 않는다. perl에서는 일반적으로 애매성이 제기될 경우에만 괄호를 리용한다. 이것을 리해하면 이 간결성을 만족하게 여길것이다. 우리가 쉘스크립를 실행하였던것과 같은 방법으로 이 프로그람을 실행시켜 보자. 이때 실지로 입력값을 누르기전에 많은 공

백을 삽입하자.

```
$ sample.pl
Enter your name:                          stallman
Enter a temperature in Centigrade:                      40.5
The temperature                             stallman
in Fahrenheit is 104.9
```

perl은 문자렬 stallman전의 공백들은 읽지만 수값 40.5전의 공백들은 읽지 않는다. perl은 기호 혹은 연산자들주위의 공백들에 대해서도 중요치 않게 여긴다.

주해

perl스크립트이름을 리용하여 perl스크립트를 실행시킬수도 있다. 이러한 경우 스크립트의 첫행에 해석기를 놓을 필요가 없다.

20.2 마지막문자를 제거하기(chop())

우의 실례에서 perl은 왜 두개의 행에 출력을 보여 주었는가? 그것은 $name에 [Enter]건에 의하여 발생된 행바꾸기문자가 포함되여 있기때문이다(쉘의 read명령문은 그렇지 않다). 그러므로 $name은 실제로는 앞의 공백을 무시한다면 ctallman\n일것이다. 많은 실례들에서 우리는 마지막문자(행바꾸기문자)를 제거해야 한다. 마지막문자의 제거는 chop()함수를 리용하여 처리할수 있다.

```
$ cat name.pl
#!/usr/bin/perl
print ("Enter your name:") ;
$name = <STDIN> ;
chop ($name) ;                # $name에서 행바꾸기문자를 제거한다
if ( $name ne " ") {
    print ("$name, have a nice day\n" );
} else {
    print ("You have not entered your name\n" );
}
```

여기서 if조건문은 그안에서 실행되는 명령문이 한개이든 여러개이든 관계없이 항상 {}를 리용해야한다. 여기서 chop는(앞으로는 함수들에 ()를 붙이지 않는다.) 마지막문자를 제거하고 그 변수에 제거(chop)된 값을 대입한다. 이것으로 하여 단 하나의 행에 출력자료를 제시할수 있다.

```
$ name.pl
Enter your name: larry wall
larry wall, have a nice day
```

hop함수를 다른 방법으로 리용할수 있다.

```
chop ($name = <STDIN>) ;             동시에 읽기 및 값주기
$lname = chop($name) ;               lname은 제거된 마지막문자를 보관한다
```

첫 명령은 C에서처럼 읽기와 제거를 한개 행으로 결합하였다. 두번째것은 문자렬의 마지막문자를 추출하는 substr함수의 특수경우이다.

참고

파일이나 건반에서 한개 행을 읽어 들일 때마다 행바꾸기문자를 고의적으로 붙이고 싶지 않다면 반드시 chop함수를 리용해야 하다는것을 잊어서는 안된다. 많은 프로그람작성자들은 이 행바꾸기문자를 chop함수로서 제거한 다음 printf에서 그뒤에 다시 행바꾸기문자를 붙이고 있다. 이것은 사실상 실용적이지 못하다.

20.3 변수와 연산자

이미 본바와 같이 perl변수들에는 형도 없고 초기화도 필요 없다. 문자렬과 수값들은 콤퓨터가 허락하는것만큼 클수 있다. 아래에는 기억해 두어야 할 몇가지 변수속성들을 보여 준다.

- 문자렬이 수값연산 혹은 비교에 리용될 때 perl은 즉시 그것을 수값으로 변환시킨다.
- 변수가 정의되지 않았으면 null문자렬로 취급하며 이 문자렬은 수값으로서 0이다.
- 문자렬의 첫 문자가 수자가 아니면 그 전체 문자렬은 수값으로서는 령으로 평가된다.

수값비교에서 perl은 awk에서와 같은 연산자들의 모임 ==, !=, >, <, >=, <=들을 사용한다. 문자렬비교에서는 쉘이 리용하는것과 비슷한 연산자들 eq, ne, gt, lt, ge, le을 리용한다. 이 연산자들의 앞에 -기호가 붙지 않는다는데 주의해야 한다. 여기서 비교는 ASCII순서맞추기렬에 따라 진행된다.

변수의 값은 실제로 모든 확장문자렬들을 포함하는 임의의 문자가 사용될수 있다. perl은 문자렬의 대소문자변환을 위한 몇가지 특수문자도 가지고 있다. 다음의 값주기는 perl변수들의 다양성을 보여 준다.

```
$x = $y = $z = 5 ;                 다중값주기
$name = "larry\t\twall\n";                  2개의 타브와 행바꾸기
$y = "A" ; $y++ ;                 이것은 B로 된다
$z = "P01" ; $z++ ;                 이것은 P02!로 된다
$todays_date = `date` ;                 지령치환을 리용한다
$name = "steve jobs" ;
$result = "\U$name\E" ;              $result는 STEVE JOBS이다
$result = "\u$name\E" ;              $result는 Steve jobs이다
```

여기에는 지금까지 보지 못하던것들도 있다. 즉 PO1을 증가시켜 PO2를 귀환시키는 기능은 perl에서만 가능하다. 확장문자렬 \U와 \u는 각각 전체 문자렬 또는 첫 문자를 대문자로 변환한다. 작용범위의 끝은 \E로 표기한다. \L과 \l은 문자렬을 소문자로 변환한다.

perl은 또한 ?와 :을 리용하여 C와 awk의 조건부값주기기능을 제공한다. 다음의 값주기는 2월달이 28일로 되여 있는지 혹은 29일로 되여 있는지를 결정한다.

```
$feb_days = $year % 4 == 0 ? 29 : 28 ;
```

perl은 또한 변수에 비교의 돌림값을 설정할수 있다. 다음의 명령문은 비교결과에 따라 $x의 값을

설정한다.

```
$x = $y == $z;
```

비교결과가 참이면 비교의 돌림값은 령이 아니다(여기서는 $y==$z). 여기서 변수 $x는 $y와 $z의 값이 같으면 1로 설정되며 그렇지 않으면 값을 가지지 않는다. perl은 0을 참의 돌림값으로 표현하는 일반적인 UNIX특징을 따르지 않는다.

련결연산자 .과 x

쉘에서와는 달리 표현식 xy(혹은 ${x}${y})는 변수의 련결(concatenaation)로 해석되지 않는다. 대신 perl에서는 변수을 련결하기 위하여 점연산자를 리용한다.

```
$ perl -e '$x=ford ; $y=".com" ; print ($x . $y . "\n");`
for.com
```

변수 $y자체가 점을 포함하므로 인용부호안에 놓아야 한다. 읽기 편리하게 하기 위하여 점의 량측에 공백을 주는것이 더 좋다.

perl은 x연산자를 문자렬을 반복시키는데 사용한다. 다음의 명령은 화면상에 40개의 별표 *를 표시한다.

```
$ perl -e 'print "*" x 40 ; '
****************************************
```

표시할 문자렬은 단일문자로 제한되지 않는다. 지어는 표현식일수도 있다. 이 연산자는 보고서들의 틀을 맞추는데서 대단히 유용하다.

20.4 문자렬처리함수

perl은 다른데서 볼수 있는 모든 문자렬함수들을 다 가지고 있다. length와 index는 16.13에서 본 기능과 같지만 substr는 아주 만능적이다. 다음의 실례에서 그것을 보기로 하자.

```
$x="abcdijklm" ;
print length($x) ;              이것은 9이다
print index($x,j) ;              이것은 5이다
substr($x,4,0) = "efgh" ;              $x를 efgh로 채워 넣는다
print "$x" ;              $x는 abcdefghijklm이다
$y = substr($x,-3,2) ;              오른쪽으로부터 추출한다
print "$y" ;              $y는 kl이다
```

index와 substr는 첫 문자의 위치를 0으로 한다. perl에서 substr는 문자렬을 추출할수도 있고 삽입할수도 있다. perl에서 substr($x, 4, 0)은 아무런 문자렬도 교체함이 없이 문자렬 $x를 efgh로 채워 넣는다. 즉 0은 교체하지 않는다는것을 의미한다. substr($x, -3, 2)는 오른쪽으로부터 세번째 위치부터 두개의 문자들을 추출한다. 이로부터 왼쪽과 오른쪽에서 위치를 지적할수 있다는것을 알수 있다. 본문의 글

자크기(대소문자)를 변경시키기 위한 네개의 함수가 있다. uc는 그의 전체 인수를 대문자로 변환시키며 ucfirst는 첫 문자만을 대문자로 변환시킨다.

```
$name = "larry wall" ;
$result = uc($name)            $result는 LARRY WALL이다
$result = ucfirst($name)            $result는 Larry Wall이다
```

함수 lc와 lcfirst는 uc계렬함수들과 반대의 기능을 수행한다. perl은 UNIX려파기들이 본문을 관리하는것과 같은 방법으로 변수들의 내용을 려과할수 있다. 후에 perl이 치환을 위하여 리용하는 두개의 중요한 함수 tr와 s를 론의한다.

20.5 지령행에서의 파일이름지정

perl은 파일로부터 자료를 호출하는 몇가지 방법을 제공한다. 아래에 파일 dept.lst를 읽기 위한 두가지 방법을 보여 준다.

```
perl -e 'print while (<>)' dept.lst
perl -e 'print <>' dept.lst            암시적인 순환
```

<>는 일반적으로 빈 파일조종자(file handle)를 의미하며 파일이름들은 지령에 인수들로 제공된다. dept.lst의 내용들은 while이 입력자료(<>)를 읽을수 있을 때까지 표시된다. perl은 순환을 의미하는 -n 선택항목도 가지고 있다.

```
perl -ne 'print' dept.lst            -en은 여기서 동작하지 않는다
```

우의 두 형식은 또한 여러개의 파일들을 련결하는데 사용할수도 있다. 아래의 형식에서 개선된 점은 지령행 그자체에서 단일한 한행조건분기를 사용한다는것이다. 아래의 지령은 동작측면에서 최소한의 grep지령과 같다.

```
$ perl -ne 'print if /wood\b/' emp.lst
5423|barry wood     |chairman  |admin    |08/30/56|160000
```

한행의 조건분기는 정규식 /wood\b를 사용한다. perl은 확장된 정규식묶음(표 20-1)을 사용하는데 여기서 \b는 단어경계를 정합하는데 사용된다. 이것은 출력에서 woodcock와 woodhouse를 제거하였다. perl의 정규식에 대해서는 후에 더 보게 될것이다.

우의 perl명령문은 스크립트내에 놓여 질수도 있다. 이때 순환이 포함되므로 해석기에 -n선택항목을 지적해야 한다.

```
# !/usr/bin/perl -n
print if /wood\b/ ;
```

우리는 하나의 제목 혹은 전체를 표시하는것과 같은 순환밖에서 몇가지 처리를 해야 할 필요가 있다. -n선택항목은 그것을 허용하지 않으므로 스크립트내에 while순환을 설정해야 한다.

```
#!/usr/bin/per1
printf ("%30s", "LIST OF EMPLOYEES\n) ;
```

```
while (<>) {
   print if /wood\b|light.*/ ;              egrep형의 표현식
}
print "\nREPORT COMPLETE\n" ;
```

우의 프로그람은 우리가 일반적으로 많이 하는것이다. 즉 내용에 앞서 제목을 출력하며 내용다음에는 어떤 결과를 표시한다.

참고

순수한 려과를 위해서는 스크립트의 시작부분에 해석기이름으로서 perl -n을 리용하시오. 이후부터는 개별적인 while순환이 필요 없다. 만일 인쇄할 제목과 꼬리부내용이 있다면 -n선택항목을 없애고 안에 while순환을 설정하시오.

20.6 기정변수($_)

앞의 프로그람들에서는 인쇄내용을 지적함이 없이 print명령을 사용하였는데 perl은 그것을 자동적으로 전체 행으로 리해하였다. perl은 **기정변수**(default variable)라고 불리우는 특수변수 $_에 입력자료로서 읽어 들인 행을 대입한다. 이것은 대단히 중요한 변수이며 간결한 코드를 작성하고 싶다면 그 변수의 기묘한 속성들을 리해해야 한다.

모든 행에 행번호를 붙여야 한다고 생각하자. 이 경우에 $_이 필요하다. 아래의 스크립트에서 설명문들은 perl에서 $_이 내부적으로 진행하는 동작을 보여 준다.

```
$ cat grepla.pl
#!/usr/bin/perl
while (<>) {                  #실제로 ($_ = <>)
     chop();                  chop($_)
     if (/From:.*\@velvet.com/) {              # if ($_=~ /From:.*\@velvet ...)
       $slno++ ;
       print ($slno . " ". $_ ."\n");
     }
}
```

여기서 변수 $_은 세 정황에서 기정변수로 동작하였다. 그 변수의 기능은 명백히 정의하기는 어렵지만 흔히 읽혀 진 마지막행이나 정합된 마지막패턴을 표현한다.

많은 함수들은 변수이름이 빠지면 변수 $_에 대하여 동작하므로 함수가 보다 작은 인수들을 가지고 사용된다 하여도 놀랄 필요는 하나도 없다. $_를 리용할수 있는가 그렇지 않은가 하는것은 항상 직관적이지는 못하며 이것은 많은 경험을 요구한다.

변수 $_은 우의 프로그람에서는 print행에 오직 한번만 나타나지만 실지로는 4개의 위치에 나타난다고 볼수 있다. <>, chop, 패턴정합은 기정적으로 변수 $_에 대하여 조작된다. print명령에서는 이 변수와 $slno을 련결시켜야 하므로 $_을 사용하였으며 그렇지 않으면 print명령은 기정적으로 $_에 대하여 조작한다. 우의 프로그람은 velvet.com령역으로부터 모든 송신자들의 전자우편주소를 찾는다.

```
$ grepla.pl $HOME/mbox
1 From: "Caesar, Julius" Julius_Caesar@velvet.com
2 From: "Goddard, John" John_Goddard@velvet.com
3 From: "Barnack, Oscar" Oscar_Barnack@velvet.com
```

$_의 값을 다시 할당할수 있다. 많은 perl함수들은 기정적으로 $_에 대하여 조작하기때문에 자기가 작업하려는 표현식으로 $_을 설정할 필요가 있다. 이것은 $_ 혹은 임의의 변수를 지적함이 없이 표현식에 대하여 perl에서 유효한 모든 중요한 함수들에 적용할수 있다. 이 모든것은 코드를 정돈되게 한다.

20.7 현재행번호($.)와 범위연산자(..)

perl은 또 하나의 특수한 체계변수 $.에 현재행번호를 보존한다. 이것은 행의 주소를 표현하는데 사용되며 파일내의 임의의 행들을 선택하는데 리용할수 있다.

```
perl -ne 'print if ($. < 4)' foo                head -3과 같다
perl -ne 'print if ($. > 7 && $. < 11)' foo                sed -n '8,10p'와 같다
```

perl은 이러한 지령들에 대한 지름법을 가지고 있다. 이때 범위연산자 ..(2개의 점)을 리용한다.

```
perl -ne 'print if (1..3)' foo
perl -ne 'print if (8..10)' foo
```

파일에서 여러개의 토막을 선택하기 위하여 여러개의 print명령문을 사용하거나 합성조건문들을 사용할수 있다.

```
if ((1..2) || (13..15)) { print ; }
```

재할당이 가능한 변수 $_와는 달리 $.은 항상 현재행번호를 유지하고 있다. 모든 입력행들이 이 변수를 통과하므로 행번호에 대하여 $.을 사용할수 있으며 $slno는 필요 없다.

20.8 목록과 배렬

perl은 목록과 배렬을 관리할수 있는 많은 함수들을 가지고 있다. 아래에 제시된것은 목록에 대한 실례이다.

```
( "Jan",123, "How are you",-34.56, Dec )
```

목록은 배렬에 할당될수 있으며 변수묶음으로도 이루어 질수 있다. 이 배렬들은 **스칼라목록**(scalar list)과 **조합배렬**(associative array)의 두가지 형식을 가진다. 여기서는 스칼라목록들에 대해 보게 된다. 다음과 같이 목록을 배렬에 대입하여 보자.

```
@month = ("Jan","Feb","Mar");                $month[0]은 Jan이다
```

이 식은 목록을 세개의 요소를 가진 배렬에 설정한다. 첫값 $month[0]은 문자렬 Jan이다. 배렬 그자체가 기호 @로 정의되였다 하여도 매개의 개별적인 요소는 $mon[n]으로 호출된다. perl에서 배렬대입은

또한 아주 유연하다. 여기에는 범위연산자를 사용할수 있으며 지어는 선택적으로 값들을 할당할수 있다.

```
@x = (1..12);
@month[1,3..5,12] = ("Jan","Mar","Apr","May","Dec") ;
```

첫 실례에서는 처음 12개의 옹근수들(1부터 12까지)을 배렬의 12개 요소들에 대입한다. 두번째 실례에서 $month[4]는 Apr로 되고 $month[2]는 이미 대입되여 있지 않으며 null값이다. 다음의 스크립트는 perl배렬의 몇가지 특징들을 설명한다.

```
$ cat ar_in_ar.pl
#!/user/bin/perl
@days_between = ("wed","Thu") ;
@days = (Mon, Tue, @days_between, Fri);              # 인용부호가 없다
@days[5,6] = ("Sat", "Sun");
$length = @days ;               # @days는 배렬길이이다
@r_days = reverse @days;                # 배렬을 반전시킨다

print ("The third day of the week is $days[2]\n") ;
print ("The days of the week are @days\n");
print ("The days of the week in reverse are @r_days\n");
print ("The number of elements in the array is $length\n");
print ("The last subscript of the array is $#days\n");
```

perl은 두번째 배렬(@bays_between)이 배렬 @days의 한 부분이 되도록 한다. perl은 또한 보조스크립트들을 선택하는 훌륭한 방법(@day s[5,6]=...)을 제공한다. revers로 배렬을 반전시킬수 있다.

배렬의 길이는 두개의 기능 $#days와 @days로 결정되게 된다. 실제상 $#days는 배렬의 마지막첨자를 보존하며 이것은 문자렬의 길이를 결정하기 위한 Korn쉘과 bash에서 사용되는 구조와 비슷하다. 배렬의 실제길이는 @days가 대입식의 오른쪽에 놓이는 경우에 거기에 보존된다. 배렬의 첨수는 0으로부터 시작되므로 $length는 $#days보다 하나 더 큰 값을 가진다.

```
$ ar_in_ar.pl
The third day of the week is Wed
The days of the week are Mon Tue Wed Thu Fri Sat Sun
The days of the week in reverse are Sun Sat Fri Thu Wed Tue Mon
The number of elements in the array is 7
The last subscript of the array is 6
```

@days는 사용되는 방법에 따라 다르게 평가된다. 즉 print @days는 모든 요소들을 표시하지만 @days가 변수에 대입되는 경우($length=@days)에는 배렬의 길이로 된다.

파일에서 배렬에로의 읽기

배렬을 채우기 위한 가장 쉬운 방법은 파일을 배렬에로 읽어 들이는것이다. 매행은 배렬의 요소가 된다.

```
@line = <>;                지령으로부터 전체 파일을 읽는다
print @line ;               전체 파일을 표시한다
```

전체 파일은 하나의 명령(@ line=< >)으로 읽혀 지며 배렬 @line의 매 요소는 행바꾸기문자를 포함한 파일의 행을 담고 있다.

주해

파일을 배렬에 읽어 들이면 모든 요소는 마지막문자로서 행바꾸기문자를 가진다. 배렬에 chop함수를 적용하여 배렬의 개별적인 요소 또는 전체요소로부터 행바꾸기문자를 제거할수 있다.

주의

perl프로그람의 시작위치에 $[= ;과 같은 설정을 하여 배렬의 첨수를 0부터가 아니라 1로부터 시작하게 할수 있다. 그러나 이것은 비표준방식으로서 첨수시작을 0으로 보는 사람들에게 혼돈을 가져다 준다.

20.9 지령행인수(ARGV[])

perl에서는 체계배렬 @ARGV[ ];에 기억되여 있는 지령행인수들을 사용한다. 첫 인수는 $ARGV[0]이다. 지령이름 그자체는 이 요소 즉 $ARGV[0]에 기억되는것이 아니라 다른 체계변수인 $0에 보관된다는것을 주의해야 한다. 다음의 프로그람은 년을 가리키는 문자렬을 인수로서 받아서 그것이 윤년인지 아닌지를 결정한다.

```
$ cat leap_year.pl
#!/usr/bin/perl
die ("You have not entered the year\n") if (@ARGV == 0 ) ;
$year = $ARGV[0] ;               # 첫 인수
$last2digits = substr($year,-2,2) ;               # 오른쪽으로부터 추출
if ($last2digits eq "00") {
    $yesorno = ($year % 400 == 0 ? "certainly" : "not") ;
}
else {
    $yesorno = ($year % 4 == 0 ? "certainly" : "not") ;
}
print ("$year is " . $yesorno . " a leap year\n");
```

첨수가 없는 @ARGV의 값은 배렬의 길이로 된다. 여기에는 보통 쓰지 않는 substr함수가 있다. 그러면 프로그람을 실행시켜 보자.

```
$ leap_year.pl
```

```
You have not entered the year
$ leap_year.pl 2000
2000 is certainly a leap year
$ leap_year.pl 1997
1997 is not a leap year
```

우의 스크립트에는 하나의 결함이 있다. 그것은 윤년검사를 위해 5개의 수값을 제시하는 경우 5번 프로그람을 실행시켜야 한다는것이다. 다음에 취급하는 foreach순환은 이러한 수고를 덜어 준다.

주해

die()는 단지 자기의 인수를 현시하고 스크립트를 완료한다. 이 함수는 파일을 열 때 혹은 잘못된 사용자입력자료를 제거할 때 오유를 처리하기 위하여 가장 많이 사용된다.

20.10 목록을 통한 순환(foreach)

perl은 목록을 리용하는 아주 쓸모 있는 순환을 위한 foreach구조를 제공한다. C쉘에서 유래된 이 구조는 아주 간단한 문법적구조를 가진다.

```
foreach $var (@arr) {
    statements
}
```

이것은 쉘의 for순환과 같이 동작한다. 배렬 @arr의 매 요소는 선택되여 변수 $var에 대입된다. 순환은 목록의 항목수만큼 계속된다. 다음의 프로그람에서는 foreach를 사용하여 몇가지 수의 2차뿌리값을 계산한다.

```
$ cat square_root.pl
#!/usr/bin/perl
print ("The program you are running is $0\n");
foreach $number (@ARGV) {               # @ARGV의 매 요소는 $number에 보존된다
     print ("The square root of $number is ". squt($number) . "\n");
}
```

배렬 @ARGV의 매 요소는 변수 $number에 대입된다. 그러면 인수로서 여러개의 수값을 가지고 스크립트를 실행시켜 보자.

```
$ square_root.pl 123 456 25
The program you are running is ./square_root.pl
The square root of 123 is 11.0905365064094
The square root of 456 is 21.3451565040625
The square root of 25 is 5
```

우의 실례에서 $number를 리용할 필요가 없다. foreach는 변수 $_에 매개 항목을 보존하며 sqrt는

그것을 가지고 작업한다.

```
foreach (@ARGV) {              $_는 기정변수이다
     print ("The square root of $_ " . sqrt() . "\n") ;
```

foreach가 오직 배렬만을 가지고 사용되는것은 아니다. UNIX지령들에 의하여 발생되는 목록과 함께 리용될수 있다. 즉 목록을 생성하는 지령대입을 리용할수 있다.

```
foreach $file (`ls`) {
```

이 순환구조는 현재등록부에서 매개의 파일을 선택하여 그것을 변수 $file에 대입한다. 우리는 이 장의 뒤부분에서 이 기능을 리용한다.

주해

perl은 for순환도 가지고 있다. 아래의 구조는 자기의 코드블로크를 세번 반복실행한다.

```
for ($i=0 ; $i<3 ; $i++) {
```

20.11 목록을 가르기(split())

perl을 사용하는 CGI프로그람개발자들은 두개의 중요한 배렬처리함수인 split와 join에 대하여 알아야 한다. split는 하나의 행 혹은 표현식을 마당들로서 분할한다. 이러한 마당들은 변수들 혹은 배렬에 대입된다. 아래에 두 함수의 문법적구조를 보여 준다.

```
($varl, $var2, $var3.... ) = split(/sep/,stg) ;
@arr = split(/sep/,stg) ;
```

split는 세개의 인수를 가지지만 보통 두개의 인수를 리용한다.

- 분리를 진행할 표현식 sep
  이것은 문자이거나 여러개의 문자로 확장될수 있는 정규식일수 있다.
- 분리될 문자렬 stg
  이것은 선택적이다. 만일 지적되지 않으면 기정적으로 $_을 사용한다.

분할하여 얻어 진 마당들은 변수 $var1, $var2, ... 혹은 배렬 @arr에 대입된다. 그러면 파일 /ect/passwd을 분석하여 권한이 없는 사용자들의 전자우편주소목록을 만드는데 첫번째 문법형식을 사용하여 보자.

```
$ cat email_create.pl
#!/usr/bin/perl -n
chop()
($uname, $password, $uid, $gid, $gcos, $home, $shell) = split (/:/,$_) ;
print "\"$gcos\"<$uname\@planets.com>\n" if ( $home =~ /\/home\// ) ;
```

$_는 split에 의해 사용되는 기정문자렬이므로 앞으로는 그것을 쓰지 않는다. 권한이 없는 사용자들은 $home을 /home/와 정합시키는것으로써 식별된다. 여섯번째 마당을 정합하기 위하여 정규식연산자 =~를 리용하였다. 주소들이 RFC822형식(13.7)으로 되여 있으므로 겹인용부호를 GCOS마당에 대하여 리

용해야 한다. "와 /은 둘 다 특수하며 은페시켜 줄것을 요구한다. 그러면 스크립트를 실행시켜 보자.

```
$ email_create.pl /etc/passwd
"george kennedy" george@planets.com
"henry blofeld" henry@planets.com
"enquiry for products" enquiry@planets.com
```

분리할 마당들이 대단히 많은 경우에는 어떻게 해야 하겠는가? 이러한 경우에는 배렬로 분리시키는 것이 더 좋다(두번째 형식). 앞의 split명령문은 배렬 @profile에 넣어 지도록 바꿀수 있다.

```
@profile = split (/:/) ;            $_는 기정문자렬이다
```

이렇게 하면 첫번째 마당은 $profile[0]에 들어 간다. 아래와 같이 배렬의 개별적인 요소들을 리용할수 있다.

```
$uname = $profile[0] ;
$gcos = $profile[4] ;            GCOS는 다섯번째 마당이다
```

대입은 $profile = $profile[3];과 같은 식으로 할수도 있다. 여기서 변수이름과 배렬이름이 서로 같아도 충돌하지 않는다. split명령문은 명백한 대입이 없이 사용될수도 있다.

```
split (/:/) ;            배렬 @_에 넣어 진다
```

우에서 보는바와 같이 split함수는 좀 더 생략되였다. 생략된 split함수는 요소들로서 $_[0], $_[1], ...을 가지는 perl의 내장배렬 @_을 채운다. 그러면 많은 프로그람들에서 리용하였던것처럼 이 형식을 리용할수 있다.

또한 split함수를 인수없이 사용할수도 있다.

```
split() ;            공백으로서 $_를 분할한다
```

여기서 행($_)은 공백으로 분리되여 배렬 @_에 넣어 진다. 이것도 만족하지 않다면 괄호까지도 없앨수 있다.

주해

split가 배렬이름이 없는 명령문으로서(같기기호 =의 오른쪽에서의 대입으로서가 아니라) 사용되면 내장배렬 @_이 리용된다. 이 배렬의 요소들은 $_[0], $_[1], ...이다. 더우기 split함수가 구분문자로서 빈 문자렬(//)을 사용한다면 문자렬의 매 문자는 개별적인 요소로서 기억된다.

20.12 목록의 결합(join)

join함수는 split와 반대의 방식으로 동작한다. 즉 모든 배렬요소들을 단일한 문자렬로서 결합한다. 첫 인수로서 구분문자를 사용한다. 나머지인수들은 배렬이름, 변수목록, 결합될 문자렬일수 있다. 아래에 매개 요일이름다음에 공백을 놓기 위한 한가지 방법을 보여 준다.

```
$weekstring = join(" ", @week_array);
$weekstring = join(" ","Mon","Tue","Wed","Thu","Fri","Sat","Sun") ;
print $weekstring ;
```

두 명령은 다음과 같은것을 출력한다.

```
Mon Tue Wed Thu Fri Sat Sun
```

이러한 결합은 대단히 중요하다. 어떤 행을 마당들로 분리하고 한두개마당을 편집한 다음 join함수를 사용하여 그것들을 다시 결합할수 있다. /etc/password의 몇개의 행들을 보기로 하자.

```
mdom:x:28:28:mailing list agent:/usr/lib/jajordomo:/bin/ksh
yard:x:29:29:yARD database admin:/usr/lib/YARD:/bin/ksh
wwwrun:x:30:65534:daemon user for apache:/tmp:/bin/ksh
fax:x:33:14:facsimile Agent:/bar/spool/fax:/bin/ksh
```

아래의 프로그람은 다섯번째 마당(GCOS)을 대문자로 바꾸기 위해서 매개 행을 마당들로 나누고 다섯번째 마당에 대하여 uc함수를 적용한 다음 반대로 그것들을 다시 결합한다.

```
#!/usr/bin/perl -n
split (/:/) ;                # @_배렬우에서 $_를 분할한다
$_[4] = uc($_[4]) ;                # 5번째 마당을 대문자로 변환한다
$_ = join(":",@_) ;                # 다시 만들어 진 행
print if (1..3) ;                # $를 인쇄한다
```

이 프로그람이 인수로서 /etc/passwd를 가지고 실행되면 처음 세개의 행이 표시된다.

```
mdom:x:28:28:MAILING LIST AGENT:/usr/lib/majordomo:/bin/ksh
yard:x:29:29:YARD DATABASE ADMIN:/usr/lib/YARD:/bin/ksh
wwwrun:x:30:65534:DAEMON USER FOR APACHE:/tmp:/bin/ksh
```

경계구분문자로 결합을 진행하는것은 프로그람작성에서 널리 리용된다. 첫번째와 마지막이름사이에는 흔히 공백이 요구된다. 또한 월, 일, 년은 -로 구분된다. 이러한 응용에 대해서는 후에 보기로 하자.

20.13 배렬내용의 수정

perl은 배렬의 내용을 관리하는 몇개의 함수들을 가지고 있다. 배렬의 시작과 끝에서 요소들을 삭제하기 위하여 perl은 shift와 pop함수들을 사용한다.

```
@list = (3..5, 9) ;                이것은 3 4 5 9 이다
shift(@list) ;                3이 제거되여 4 5 9로 된다
pop (@list) ;                마지막요소를 제거하여 4 5로 된다
```

unshift와 push함수는 배렬에 요소를 추가한다. 이전 실례의 끝에서 @list의 나머지값들에 대하여 이 함수들을 적용하여 보자.

```
unshift(@list, 1..3) ;                1,2,3이 추가되며 결국 1 2 3 4 5 로 된다
push (@list, 9) ;                끝에 9가 추가된다. 즉 1 2 3 4 5 9
```

splice함수는 우와 같은 네개의 함수들이 할수 있는 모든것을 다 할수 있다. 추가적으로 배렬의 임의

의 위치에 요소들을 추가하거나 삭제하기 위하여 네개의 인수까지 사용한다. 두번째 인수는 삽입 혹은 삭제를 어디서부터 시작하겠는가 하는 편위(offset)값이다. 세번째 인수는 제거될 요소의 개수를 표시한다. 이것이 0이라면 요소들은 추가되여야 한다. 새로 교체될 목록은 네번째 인수에 의하여 지적된다.

```
splice (@list, 5, 0, 6..8) ;            6번째 위치에 추가된다. 즉 1 2 3 4 5 6 7 8 9
splice (@list, 0, 2) ;            시작위치로부터 제거한다. 즉 3 4 5 6 7 8 9
```

어떤 스크립트를 개발하여 배렬처리함수들에 우리의 지식을 결합해 보자. 이 스크립트는 IP주소(202.54.9.1와 같은)를 인수로서 받아서 그것을 2진수로서 변환한다.

```
print "The IP address in binary is" ;
foreach $number (@_) {
    $original_number = $number ;
    until ($number == 0) {
        $bit = $number % 2 ;              # 나머지비트를 찾는다
        unshift (@bit_arr,$bit) ;              # 시작위치에 비트를 삽입
        $number = int($number / 2 )
    }
    $binary_number = join ("",@bit_arr) ;              # 아무것도 련결하지 않는다
    substr($binary_number,0,0) ="0" x (8 - length($binary_number)) ;
    print ("$binary_number") ;
    splice(@bit_arr,0,$#bit_arr+1) ;              # 모든 배렬요소들을 제거한다
}
print chr(10) ;              # 행바꾸기를 인쇄한다
```

여기서 split는 IP주소를 네개의 마당들로 분리하여 그것들을 배렬 @_에 보관한다. 10진수를 2진수로 변환하자면 상을 2로 계속 나누어 가며 그 나머지를 모두 모은 다음 그것을 반대로 다시 모아야 할것이다. unshift는 반전처리를 수행한다.

substr는 IP주소의 매개 8진수가 8비트로서 표시되도록 시작부분에 여분의 0을 배치한다. foreach의 순환본체는 매 8진수를 2진수로서 표시한다. splice는 다음반복이 시작되기전에 배렬을 가득 채운다. 그러면 스크립트를 실행시켜 보자.

```
$ ipadd2binary.pl 224.67.34.06
The IP address in binary is 11100000 01000011 00100010 00000110
```

이 프로그람을 부분망마스크(23.1)를 변환하는데도 사용할수 있으며 그때 두개의 주콤퓨터가 같은 부분망에 속하는지 아닌지를 검사할수 있다.

20.14 조합배렬

perl은 배렬의 또 다른 형태 즉 **조합배렬**(associative array)도 리용한다. 이 배렬은 반점으로 구분되는 련속적인 값들에서 배렬의 첨자와 값이 엇바뀌여 놓인다. 실례로 조합배렬 %region은 다음과 같이 정의될수 있다.

```
%region = ("N","North","S","South","E","East","W","West") ;
```

이 배렬은 그 이름앞에 기호 %를 리용한다. 이 대입식은 배렬정의에서 값의 앞에 첨자를 가진 네개의 요소로 이루어 진 배렬을 만든다. 문자렬로 되여 있는 배렬의 첨자는 [ ]가 아니라 {}로 둘러 막아야 한다. 실례로 $region{"N"}은 값 North를 가리킨다. CGI프로그람개발자들은 조합배렬을 잘 알아야 한다.

다음의 프로그람은 %region배렬을 리용하여 어떤 구역들에 대한 코드를 확장한다. 여기서는 두개의 조합배렬함수들 Keys와 values를 어떻게 사용하는가에 대해서도 보여 준다.

```
$ cat region.pl
#!/usr/bin/perl
%regin = ("N","North","S","South","E","East","W","West") ;
foreach $letter (@ARGV) {
   print ("The letter $letter stands for $region{$letter}" . "\n );
}
@key_list = keys(%region) ;                # 첨자들의 목록
print ("The subscripts are @key_list\n" ;
@value_list = values %region ;                # 값목록
print ("The values are @value_list\n");
```

keys는 개별적인 배렬(여기서는 @Key_list)에 첨자목록을 보존하며 values는 또 다른 배렬(여기서는 @valne_list)에 매 요소의 값을 보유한다(여기서는 괄호를 리용하지 않았다). 스크립트에 한쌍의 단일문자들의 렬을 제공하는것으로써 검사하여 보자.

```
$ region.pl S W
The letter S stands for South.
The letter W stands for West
The subscripts are S E N W
The values are South East North West
```

이것은 중요한 의미를 포함하고 있다. 조합배렬에서 열쇠와 그의 값을 개별적으로 둘 다 추출할수 있다. 또한 이 값들을 set명령문이 모든 환경변수들을 보여 주는것과 같은 방법으로 현시할수 있다.

```
foreach $key (keys %region) {
    print "$key" . "=" . "$region{$key}\n" ;
```

우의 프로그람은 매개 열쇠에 대한 값을 찾아 내서 그것을 차례로 변수 $key에 보관한다. 다음 그것

을 %region의 첨자로서 사용할수 있다. 요소 $region{$key}는 아래에서 아래에서 볼수 있는것처럼 매개 열쇠에 대한 값을 보존한다.

```
S=South
E=East
N=North
W=West
```

보통 keys는 열쇠문자렬을 우연수렬로서 되돌려 준다. 목록을 자모순으로 정돈하기 위해서는 keys 함수와 함께 sort함수를 리용해야 할것이다. 정방향정렬과 역방향정렬을 다 가질수 있다.

```
foreach $key (sort(keys %region)) {
@key_list = reverse sort keys %region ;                 ()는 없다.
```

주해

perl의 내장배렬 %ENV는 쉘의 모든 환경변수들을 다 보관하고 있다. 실례로 $ENV{'PATH'}는 쉘의 $PATH의 값을 포함한다. 여기서 론의한 기술을 리용하여 이러한 변수들을 쉽게 호출할수 있다.

출현회수의 계수

조합배렬은 항목의 출현회수를 계산할 때 아주 쓸모 있다. 실례자료기지에서 직위에 따르는 인원수를 보여 주는 보고서를 작성한다고 하자. 앞에서 awk와 기본적인 UNIX려파기들로써 이와 같은 류사한 실습을 해보았다. 그러면 perl로 이것을 해보자. 프로그람작성언어에 대하여 이것은 일정한 작업량으로 되지만 perl에서는 작은 코드량으로서도 이것을 수행할수 있다.

```
$ cat count.pl
#!/usr/bin/perl
while (<>) {
    split (/\|/) ;                # |는 의미해제되여야 한다
    $dept = $_[3] ;               # 직위는 네번째 마당이다
    $deptlist{$dept} += 1;             # ++와 같다
}
foreach $dept (sort (keys %deptlist)) {
    print ("$dept: $deptlist{$dept}\n") ;
}
```

프로그람은 두개의 부분으로 구분된다. 첫번째 부분의 while구조는 읽혀 진 매행에 대하여 $dept의 값을 려과하며 배렬 $deptlist의 개별적인 요소의 계수값을 증가시킨다. 모든 입력자료가 다 읽혀 진후에 foreach구조는 %deptlist로부터 변수 $dept에로 매개 열쇠값을 대입한다. $deptlist{$dept}는 매개 열쇠에 대한 축적된 총량으로 된다. 다음의 정렬된 출력은 perl의 성능을 보여 준다.

```
$ count.pl emp.lst
accounts   : 2
admin      : 1
```

```
marketing  : 4
personnel  : 2
production : 2
sales      : 4
```

perl코드의 몇개의 행에 의하여 직위별로 그 인원수를 목록으로 보여 주었다.

20.15 정규식과 치환

perl은 UNIX체계(POSIX에 의하여 지적되는 표현식을 제외한)에서 볼수 있는 최상의 모든 가능한 정규식들을 제공한다. 이것들은 이미 패턴정합에 대하여 취급할 때 리용하였다. perl은 grep와 sed에서만 리용되는 표현식은 물론 egrep와 awk에 의하여 리용되는 정규식을 모두 인식한다. 또한 perl은 자기자체의 표현식들도 가지고 있다(표 20-1).

표 20-1. perl이 리용하는 정규식들

기 호	의 미
\w	단어문자를 정합한다([a-zA-Z0-9]와 같다)
\W	단어문자를 정합하지 않는다([^a-zA-Z0-9]와 같다)
\d	수값을 정합한다([0-9]와 같다)
\D	수값을 정합하지 않는다([^0-9)와 같다)
\s	공백문자를 정합한다
\S	공백문자를 정합하지 않는다
\b	단어경계를 정합한다
\B	단어경계를 정합하지 않는다

20.15.1 s와 tr함수들

s와 tr함수들은 perl에서의 모든 치환을 처리한다. s함수는 sed에서의 s지령과 같은 방법으로 사용된다. tr는 UNIX의 tr지령이 하는것과 같은 방법으로 문자들을 처리하지만 약간 다른 문법적형식을 가진다. 아래에 $_에 대한 이 함수들의 리용방법을 보여 준다.

```
$ cat substitue.pl
#!/usr/bin/per ?n
s/\|/:/g ;                # |는 의미해제되여야 한다
tr/a-z/A-Z/;
s#/#-#g ;               # 구분문자는 #로 된다
s/ +:/:/g ;               # 구분문자앞에 있는 여러개의 공백들을 압축한다
print if (1..3) ;
```

여기서 다섯가지 모든 동작은 $_에 대하여 수행된다. 첫번째 s함수는 |을 :로 교체하며 tr는 모든 글자를 대문자로 바꾼다. 두번째 s함수는 /기호모두를 -로 바꾼다. 여기서 교체될 기호 /은 의미해제되지

않기때문에 구분문자를 /으로부터 #으로 변경하였다.

perl명령문은 sed에서와 같이 바로 앞의 동작의 결과로 얻어 진 행에 대하여 동작한다. 첫번째 s함수에 의하여 |이 :으로 바꾸어 지므로 마지막 s함수는 :를 호출할수 있다. 여기서 s는 :의 왼쪽에 있는 공백을 지우기 위하여 egrep형식의 메타문자 +를 사용한다. 출력결과는 다음과 같다.

```
$ substitute.pl emp.lst
2233:CHARLES HARRIS:G.M.:SALES:12-12-52: 90000
9876:BILL JOHNSON:DIRECTOR:PRODUCTION:03-12-50:130000
5678:ROBERT DYLAN:D.G.M.:MARKETING:04-19-43: 85000
```

두 함수가 다 역시 변수들에 대하여 동작한다. 이 경우 정합을 수행하기 위하여 연산자 =~를 리용해야 하며 그것을 반대로 하기 위해서는 !~를 리용해야 한다.

```
$line =~ s/:/-/g ;              $line은 재할당된다
$line =~ tr/a-z/A-Z ;             여기서도 같다
```

s와 tr에서 기발들을 리용할수 있다. s에서 g기발을 리용하여 전역치환을 진행할수 있으며 교체된 패턴이 표현식으로 평가된다는것을 가리키기 위하여 e기발을 리용할수 있다. tr는 기발로써 UNIX의 모든 tr선택항목들을 사용한다. 즉 s는 여러번의 출력을 압축하고 c는 문자를 완성하며 d는 문자를 삭제한다(9.13).

substitute.p1에서 마지막 s함수는 공백대신에 \s를 리용할수 있다.

```
s/\s+:/:/g;
```

perl은 공백, 수자, 단어경계들을 표현하기 위한 몇가지 의미해제된 문자들을 제공한다(표 20-1). 이 문자들을 리용하여 정규식을 간결하게 할수 있다.

\s ----공백문자

\d ----수자

\w ----단어문자

이 의미해제된 문자모두는 소문자로 된것의 거꿀표현인 대문자로 된 짝을 가진다. 그러므로 \D는 수자가 아닌 문자이다. 앞에서 이미 단어를 경계로 하여 패턴을 정합하기 위하여 정착문자렬 \b를 리용해 보았다(20.5). perl에서 사용하는 정규식의 완전한 목록을 부록 3에 보여 준다.

20.15.2 IRE와 TRE기능

perl은 {}와 [ ]가 의미해제되지 않는다는것을 제외하고는 grep와 sed에서 리용되는 IRE와 TRE도 사용한다(15.12) 실례로서 아래에 문자수가 512를 벗어 나는 긴 행을 탐색하는 방법을 보여 준다.

```
perl -ne 'print if /.{513,}/' foo              {와 }전에 \이 없다
```

TRE기능을 사용한 \d의 중요한 응용에 대해 고찰하여 보자. IP주소들은 네개의 십진수를 사용하며 몇개의 구성파일들에서 자기의 망주소를 192.168.x.x으로부터 172.16.x.x으로 변경하고 싶은 경우에 꼬리표기능이 대단히 유용하게 리용된다.

```
s/192.168.(\d+).(\d+)/172.16.\1.\2/g ;
```

(\d+)는 sed의 \([0-9] [0-9]*\)와 비교해 볼 때 더 간결한 형식으로 십진수들의 묶음을 만든다. 여기에는 문자(.)에 의하여 구분되는 두개의 10진묶음이 있다. 이 묶음들은 교체될 문자렬에서 \1과 \2로서 표현된다.

추가된것으로서 perl은 다음번의 묶음화가 진행될 때까지 묶음화된 패턴들을 기억하는 변수 $1, $2, ...을 리용한다. 프로그람에서 그것들을 후에 재호출하게 된다.

```
if (/(\d+)\.(\d+)\.(\d+)\.(\d+)/) {
   if ($1<127) {
       print "Host Address is $2.$3.$4\n" ;
   }
   elsif ($1 < 192) {
       print "Host Address is $4\n" ;
   }
   else {
       print "Host Address is $4\4" ;
    }
}
```

이 프로그람은 IP주소를 가지고 있는 행을 찾아서 그의 네 요소들을 추출한다. 다음 주소가운데서 주콤퓨터주소를 결정하기 위하여 23.1에서 론의된 규칙들을 적용한다. 묶음화된 패턴령력은 현재패턴 이상으로 확장하며 다음의 명령들에 쉽게 사용될수 있다. 이 프로그람을 시작위치의 변수 $_에 IP주소를 대입하여 검사해 보시오.

주해

\d는 십진수를 표현하며 \s는 공백문자, \w는 단어문자, \b는 단어경계로서 패턴을 정합한다. 이것들과 짝을 이루는 대문자기호들은 소문자의 경우와 반대기능을 수행한다.

20.15.3 파일의 직접편집

perl은 표준출력이나 개별적인 파일에 쓰기하는것대신에 입력파일 그자체를 편집하고 재쓰기할수 있다. sed에서는 출력을 림시파일로 절환하고 다음 그것을 반대로 원래의 파일로 절환한다. 파일들의 묶음에 대해서는 for순환을 사용하였다. perl에서는 그렇게 하지 않고 i선택항목을 리용하여 여러개의 파일들을 그 자리에서 **직접편집**(in-place editing)할수 있다.

```
perl -p -i -e "s/<B>/<STRONG>/g" *.html *.htm
```

이 명령은 모든 HTML파일들의 모든 행에서 꼬리표 <B>를 <STRONG>으로 바꾼다. 파일 그자체들은 새로운 출력자료로서 덧쓰기된다. 직접편집이 모험적인것이라고 생각된다면 이 동작을 진행하기전에 파일들을 여벌복사해 놓을수 있다.

```
perl -p -i.bak -e "tr/a-z/A-Z/" foo1 foo2 foo3 foo4
```

이것은 먼저 foo1파일을 foo1.bak로, foo2를 foo2.bak 등 이러한 식으로 파일을 여벌복사한 다음 매개 파일들의 소문자자체를 대문자로 변환한다.

20.16 파일처리

지금까지는 UNIX의 지령행에서 입력파일이름을 지적하였다. perl은 스크립트 그자체에서 자료흐름의 원천지와 목적지를 경코드화(hard-code)하게 하는 저준위파일처리함수들도 제공한다. 파일은 아래와 같이 읽기방식으로 열려 진다.

```
open (INFILE,"/home/henry/mbox") ;                인용부호를 붙여야 한다
```

여기서 INFILE은 파일 mbox에 대한 파일조종자(filehandle)이다(경로이름이 사용되지 않으면 파일이 현재등록부에 있다고 가정한다). 앞으로 perl명령문들은 파일을 호출할 때 파일이름이 아니라 파일조종자를 사용하게 된다. 여기서 주의할 점은 파일이름을 바꾸면 그 즉시에 파일조종자의 정의를 변경시켜야 한다는것이다.

파일은 자기의 원래의 의미를 가지는 쉘에서와 같은 연산자 >와 >>에 의하여 쓰기방식으로 열려 진다.

```
open (OUTFILE, ">rep_out.lst") ;
open (OUTFILE, ">>rep_out.lst") ;
```

perl의 파일조종자들은 관흐름과도 결합될수 있다. 쉘프로그람작성자들에게 있어서 아래의 명령문들의 의미는 아주 명백하다.

```
open (INFILE, "sort emp.lst |") ;              sort출력으로부터 입력을 얻는다
open (OUTFILE, "| lp");                  인쇄완충기에로의 출력
```

다음의 스크립트는 경코드화된 입력 및 출력파일이름들을 가지고 있다. 또한 마지막에 파일들을 닫는다.

```
$ cat rw.pl
#!/usr/bin/perl
open (FILEIN, "desig.lst") || die ("Cannot open file");
open (FILEOUT, ">desig_out.lst" ;
while (<FILEIN>) {              # 파일에 행들이 있는 동안
     print FILEOUT if($. < 4 );              # if (1..3)도 리용할수 있다
}
close (FILEIN);
close (FILEOUT);
```

명령문 while (<FILEIN>)은 FILEIN파일조종자에 의해 표현되는 파일로부터 한번에 한행씩 읽어서 변수 $_에 보존한다. <FILEIN>명령문이 실행될 때마다 다음행이 읽혀 진다. 다음과 같은 방법으로 단 하나의 행만을 읽고 출력할수 있다.

```
$_ = <FILEIN>;               $_에 할당된다
print ;                print는 기정적으로 $_을 리용한다
```

print는 인수로서 파일조종자를 사용한다. 파일조종자를 지적하지 않으면 출력이 그 파일에 씌여 진

다. 스크립트 rw.p1에서 print명령문은 $_를 직접 FILEOUT라는 파일조종자가 할당되여 있는 파일 desig_out.lst에 쓰기한다.

스크립트를 완료하기전에 파일들을 닫지 않는다고 해도 perl은 자체로 파일을 다 닫아 준다. close명령문은 프로그람에서 후에 파일을 다시 열려는 경우에 지시자를 파일의 첫 위치에로 옮겨 놓는다. 인수없이 스크립트를 실행시키면 출력이 말단으로 가는것이 아니라 파일 desig_out.lst로 간다.

참고

몇개의 print명령문들이 같은 파일조종자(례를 들어 FILEOUT)에 쓰기해야 한다면 명령문 select(FILEOUT)를 리용하여 기정적인것으로서 이 파일조종자를 할당할수 있다. 이 경우 print명령문들에 FILEOUT인수를 리용할 필요가 없다.

20.17 파일검사

perl은 정교한 파일검사체계를 가지고 있다. 이것은 Bourne쉘의 능력을 초월하며 지어는 몇가지 방법에서 find지령을 릉가한다. 다음의 명령들은 파일의 가장 일반적인 속성들중 일부를 검사한다.

```
$x = "rdbnew.lst" ;
print "File $x is readable\n" if -r $x ;
print "File $x is executable\n" if -x $x ;
print "File $x has non-zero size\n" if -s $x ;
print "File exists\n" if -e $x ;
print "File $x is a text file\n" fi -T $x ;
print "File $x is a binary file\n" if -B $y ;
```

perl의 파일검사는 좀더 나아 가서 파일의 변경 및 접근시간들을 아주 정확히 알려 줄수 있다. 다음의 스크립트는 C나 awk의 printf명령문을 표준형식으로 사용하고 있다. 여기서는 2시간 40분전에 변경된 파일들을 다 탐색한다.

```
$ cat when_last.pl
#!/usr/bin/perl
# Finds out files less than 2.4 hours old
foreach $file(`ls`) {
   chop ($file) ;
   if (($m_age = -M $file) < 0.1) {     #tenth of a day i.e., 2.4 hours
       printf "File %s was last modified %0.3f days back \n", $file, $m_age ;
   }
}
```

-M $file은 $file이 마지막으로 변경된 때로부터 현재까지 지나온 시간을 되돌려 준다. 이것은 C로부터 유래된 perl의 일반적인 기능으로서 검사(<0.1) 및 대입($m_age = ...)을 동시에 진행할수 있다. 아래의 출력결과를 보시오.

```
$ when_last.pl
File bf2o.sh was last modified 0.063 days back
File profile.sam was last modified 0.082 days back
File when_last.pl was last modified 0.000 day back
```

우에서 마지막파일이 금방 변경된것처럼 보이지만 소수점아래 세자리로서는 충분하지 못하다. 정확한 시간이 요구되면 printf형식의 길이를 증가시켜야 한다. perl은 파일속성검사외에도 파일이나 등록부를 매우 쉽게 관리할수 있다. 그러한 지령들로서 chmod, chown, chgrp,chdir(cd와 같다.), mkdir, rmdir, rename(mv와 같다.), link, unlink(rm과 같다.), umask가 있다. 파일조종자와 마찬가지로 여기서도 등록부파일조종자(directory filehandle)를 써서 등록부를 열수 있다.

20.18 부분루틴

perl은 부분루틴들로서 값을 되돌리는 수속과 함수를 관리한다. 부분루틴들은 그 이름앞에 기호 &을 붙여서 호출된다. 부분루틴의 인수들은 배렬 @_에 보존된다. 부분루틴내의 변수들은 그것들이 이 부분루틴을 호출한 프로그람에서 리용할 필요가 없는 경우에는 국부변수로 선언될수 있다.

많은 응용프로그람들에서는 사용자이름과 통과암호를 입력할것을 요구한다. 이것은 같은 코드를 두번 실행시키는것이므로 부분루틴으로 쓸수 있는 아주 좋은 후보이다. 다음의 프로그람에서는 부분루틴 take_input()를 사용한다. 여기서는 인수로서 프롬프트문자렬을 받아서 단어문자들에 대한 입력자료를 검사한다. 다음 입력된 값을 되돌려 준다.

```
$ cat input.pl
$!/usr/bin/perl
system ("tput clear") ;                # UNIX지령을 실행한다
$username = &take_input ("Oracle user-id:") ;
$password = &take_input ("Oracle password:", "noecho") ;
print "\nThe username and password are $username and $password\n" ;
systerm ("sqlplus $username/$password @query.sql >/dev/null") ;

sub take_input {
   local ($prompt, $flag) = @_;                # @_는 부분루틴의 인수들을 보유한다
   while (1) {               #(1)은 늘 참이다
     print ("$prompt") ;
     system("stty -echo") if (@_ == 2) ;               # Echo방식이 아니다
     chop ($name = <STDIN>) ;
     system("stty echo") if (@_ == 2) ;               # Echo방식이다
     last if $name =~ /\w/ ;               # $name이 적어도 한개의 단어문자를 가지면
   }               #순환에서 탈퇴한다
```

```
    return $name ;
  }
```

부분루틴의 인수들은 체계배렬 @_에 넣어 지며 국부변수인 $prompt와 $flag에 다시 대입된다. 부분루틴에서 검사되는것은 보내진 인수들의 수(@-==2)이다. 두개의 인수가 보내지면 UNIX의 stty지령은 통과암호입력의 표시를 *로 한다.

perl에서 last명령문은 쉘의 break명령문 즉 순환을 완료하는 명령문과 같은 기능을 수행한다(perl에서는 continue대신에 next명령문을 리용한다). 여기서는 입력에 적어도 한개의 단어문자가 있으면 순환이 중지된다. 아래에 통과암호가 표시되지 않는다는것을 확인할수 있는 결과를 보여 준다.

```
$ input.pl
Oracle user-id: !@#$%^&*                단어문자가 아니다
Oracle user-id: scott
Oracle password: *****                  통과암호는 표시되지 않는다
The username and password are scott and tiger
    ......Executes SQL*Plus script query.sql........
```

자주 사용되는 부분루틴들은 개별적인 파일들에 보관하여야 한다. 시작위치에 require명령을 배치하여 부분루틴을 포함하는 파일을 프로그람에서 읽어 들이게 할수 있다. 파일 oracle_lib.p1에 take_input 부분루틴을 보존하였다면 다음의 두가지 작업을 해야 한다.

- perl해석기를 지정한 다음에 require "orade-lib.p1"명령문을 삽입하시오.
- 하나이상의 부분루틴을 포함하는 파일의 끝에 명령문 1;을 놓으시오. perl문서는 필요한 파일에 대하여 그끝에 참의 값을 놓을것을 요구한다. perl에서 0이 아닌 임의의 값이 참의 값이므로 1;은 참을 되돌려 준다. CGI프로그람작성을 취급하는 절에서 이 기능들을 론의하게 된다.

주해

우의 실례에서 $prompt대신 $_[0]을, $flag대신 $_[1]을 놓을수 있다. 변수들을 부분루틴밖에서도 볼수 있게 하자면 부분루틴의 인수들을 대입할 때 단어 local을 없애야 한다.

20.19 결론

이 장은 한가지 응용에 대한 매우 방대하고 밀집된 내용을 서술하고 있다. 그러나 아직도 설명할것은 대단히 많다. perl은 여기에 없는 망작업 혹은 프로쎄스사이의 통신과 관련되는 특수함수들을 가지고 있다. 그의 객체지향적인 도구들과 기술들은 여기서 무시하였다. perl은 sed프로그람을 perl로 변환하는 s2p와 awk프로그람을 perl로 변환하는 a1p라는 두개의 유용한 도구를 가지고 있다.

UNIX의 《정신》은 perl에 기본적으로 반영되여 있다. 중요하고도 매력 있는 UNIX기능들에 대하여 다시한번 생각해 보시오. 이러한 기능들은 모두 perl에 있다. perl은 UNIX의 자랑이라고 말할수 있다.

20.20 perl에 의한 CGI프로그람작성

자기의 Web상의 양식에 자료를 입력하고 종속단추를 누르면 이때 열람기는 다른쪽에 있는 Web봉사기에로 양식자료를 전송한다. Web봉사기는 자기자체에 이 자료들을 처리할수 있는 고유한 기능이 없으므로 외부응용프로그람에 그것을 보내게 된다. 이 응용프로그람은 받은 자료에서 내용을 추출하며 어떤 일감을 처리한다. 즉 자료기지에서 어떤 자료를 추가, 수정, 삭제하거나 또 어떤 자료에 대하여 질문하며 그의 탐색결과를 반대로 보낼수 있다. Web봉사기는 정보를 넘겨 주거나 또 받기 위한 CGI(Common Gateway Interface)응용프로그람에 대한 련결부로서 동작한다.

CGI프로그람은 변수들을 그의 값들로부터 분리한다든가 그리고 부호화된 문자들을 ASCII문자로 변환하는것과 같은 양식자료에 대한 려과를 할 필요가 있다. 흔히 이 프로그람은 자기의 모든 꼬리표들을 가지고 HTML문서를 발생시켜야 하며 열람기에 그 문서를 보내야 한다. 이 프로그람은 C, Java, 쉘 혹은 perl과 같은 임의의 언어로 씌여 진다.

나머지절들에서 우리는 두가지 처리 즉 양식자료를 려과하고 HTML을 만들기 위하여 perl을 리용한다. 여기서 우리는 작은 프로그람을 개발하는데 필요한 몇가지 perl기능들을 더 볼것이다.

20.20.1 HTML양식에 대한 리해

우리는 CGI프로그람에 앞에서와 류사한 종업원자료기지(15.1)를 사용한다. 그러나 여기서는 Web열람기로부터 그것을 호출한다. 작업에 착수하기전에 HTML의 몇가지 꼬리표들의 의미를 리해할 필요가 있다. 그래야만이 CGI perl프로그람이 열람기에 자료를 보낼 때 정확한 꼬리표들을 만들게 할수 있다.

그림 20-1. Netscape에 의한 HTML양식(emp_form.html)

그림 20-2에 보여 주는 HTML코드는 <form>끄리표로써 양식을 지적한다. 이 끄리표의 action속성은 봉사기에서의 perl프로그람(emp_add.pl)을 가리키는 URL을 지적한다. 이 프로그람은 자료기지에 한 행을 추가한다. 사용자입력을 접수하는 이 양식을 그림 20-1에 보여 주었으며 그의 코드는 그림 20-2에 제시한다.

```
<html><head>
   <title>The Employee Database</title>                          제목띠에 나타난다
</head<
<body>
<h1> Employee Form </h1>                                   굵고 큰 폰트로 나타난다
<hr>
<form action="http://localhost/cgi-bin/emp_add.pl"method=get>
        Emp_id: <input type="text" name="empid" size=4> <br>
          name: <input type="text" name="ename" size=30> <br>
   Designation: <input type="text" name="desig" size=15> <br>
    Department: <input type="text" name="dept" size=15> <br>
 Date of birth: <input type="text" name="dtbirth" size=10> <br>
        Salary: <input type="text" name="salary" size=10> <br>
<center>
<input type=submit value="Add">                              Add단추는 중심에 놓인다
</center>
</form>
</body>
</html>
```

그림 20-2. emp_form.html:form문서에 대한 HTML코드

모든 HTML문서는 머리부코드(첫 다섯개의 행)와 끄리부코드(마지막 2개의 행)로 이루어 진다. perl은 대부분의 CGI응용프로그람들에서 이러한 행들을 만들어야 하므로 여기서는 이 프로그람들에서 사용되는 두개의 부분루틴을 만든다.

HTML문서의 본체는 <form>과 </form>이라는 끄리표들로서 둘러 막힌 단순한 양식으로 구성된다. 여기에는 종업원자료기지에서 여섯개의 마당을 접수하는 여섯개의 본문칸들이 있다. 이 마당들에 입력되는 값들은 empID, ename, desig 등과 같이 그와 대응하는 변수이름들과 짝을 이룬다.
끄리표는 매개 본문칸이 개별적인 행에 놓여 지도록 한다.

<form>끄리표의 action속성은 국부주콤퓨터(localhost) 그자체의 URL을 지적한다. 이 URL은 등록부 /cgi-bin의 emp_add.p1이라는 perl프로그람을 지적하며 이 등록부는 CGI프로그람들이 일반적으로 들어 있는 곳이다. 이 프로그람은 Add이라는 표식이 붙어 있는 종속형의 단추가 마우스에 의해서 눌리워 졌을 때 실행된다.

20.20.2 질문문자렬

이제는 알수 있는것처럼(14.10) 열람기는 자기의 요청머리부를 통하여 봉사기에 자료를 보낸다. 양식자료가 어떻게 구성되는가를 리해하기 위하여 이름 empID, ename, desig(<input>끄리표의 name속성)

를 가진 세개의 마당만을 고찰해 보자. 이 세개의 마당들에 각각 1234, henry higgins, actor라는 값을 입력하자 그러면 열람기는 **질문문자렬**(query string)안에 <이름=값>의 형식으로서 전체 자료를 다음과 같이 문자렬화한다.

```
empid=1234&ename=henry+higgins&desig=actor
```

이 단일한 문자렬은 URL에서 정의한 봉사기에 보내진다. 여기서 &는 구분문자로서 작용한다. 또한 열람기는 공백문자를 +로서 부호화하였다. 이 자료를 리용하자면 perl은 문자렬을 두번 분해해야 한다. 즉 첫번째는 매개의 <이름=값>형식의 쌍들을 추출하는것이고 다른 하나는 이름과 값을 분리하는것이다. 이것은 지적된 메써드에 따라서 두가지 방법으로 진행되는데 이러한 내용은 다음에 취급한다.

20.20.3 요청메쏘드 GET와 POST

<form>꼬리표는 또 다른 속성 즉 메쏘드를 가지고 있다. 이것은 자료가 봉사기에 보내지는 방법을 표현한다. 일반적으로 우에서 본 질문문자렬은 두가지 방법으로 보내진다.

- GET-이 메써드는 구분문자로서 ?를 사용하여 URL에 질문문자렬을 추가한다. URL은 질문문자렬과 함께 다음과 같이 나타난다.

  ```
  http://localhost/cgi-bin/emp_add.pl?empid=1234&ename=
  henry+higgins&desig=actor
  ```

  봉사기는 요청머리부에서 get명령문을 분석하여 ?다음의 자료를 자기의 환경변수 QUERY_STRING에 기억시킨다. 이 변수는 임의의 CGI프로그람에 의해서 사용될수 있다.
- POST-이 메써드와 함께 열람기는 먼저 질문문자렬을 보내기에 앞서 문자렬에 포함되여 있는 문자의 수를 먼저 봉사기에 보낸다. 봉사기는 이 수값을 CONTENT_LENGTH변수에 기억시킨다. 그리고 CGI프로그람의 표준입력으로서 그 문자렬을 제공한다. perl은 read함수로서 이 자료를 CONTENT_LENGTH에 의해 정의된 수만큼 읽는다.

메써드 그자체는 봉사기환경에서 REQNEST_METHOD로서 쓰인다. 우리의 HTML실례양식에서는 메써드로서 GET를 사용한다. GET메써드에 대해서는 문자렬의 크기가 1024로서 제한된다. 많은 자료를 보내려면 POST를 사용해야 한다. 그러나 질문문자렬의 구조는 두 경우에 다 같으며 emp_add.p1프로그람은 두개의 메써드를 리용한 자료를 다 조종할수 있어야 한다. 이 실례에서 POST보다 GET를 먼저 선택할 리유는 없다.

20.21 양식자료의 처리

CGI프로그람 emp_add.p1은 QUERY_STRING(GET용) 혹은 STDIN(POST용)에서 자료를 분석하여야 한다. 다음 추출된 자료를 한행으로 결합하며 그것을 자료기지의 능동인 본문파일에 추가하여야 한다. 더 잘 리해하기 위하여 우리는 열람기창문에 중요한 CGI환경변수들의 내용을 출력할것이다. CGI프로그람은 이러한 통보문을 표시하는데 요구되는 HTML을 만들수 있어야 한다.

20.21.1 머리부와 꼬리부를 위한 부분루틴들의 만들기

모든 HTML문서들이 공통적인 머리부와 꼬리부토막을 가지고 있으므로 그것들을 위한 두개의 부분

루틴들을 먼저 만들어 보자. Htmlheader는 머리부를 현시한다.

```
sub Htmlheader {
    local ($title,$h1) = @_ ;
     print << "MARKER";                here문서
     <html>
     <head>
     <title>$title</title>                변수치환
     </head>
     <body>
     <h1>$h1</h1>
MARKER
}
```

Htmlheader는 $title과 $h1에 보존될 두개의 인수를 받는다. 이것은 부분루틴을 호출하면 제목과 첫준위머리부(first level header)를 지적하기 위한 선택권한을 가지게 된다는것을 의미한다. 여기서 우리는 here문서처럼 하나의 printf명령을 사용하였다. 표식자끄리표(marker tag)를 겹인용부호로 둘러 막는것은 변수치환이 가능하도록 하기 위해서이다.

끄리부부분루틴은 더 간단하다.

```
sub Htmlfooter {
print "</body>\n</html>\n" ;
}
```

이 두개의 부분루틴들을 개별적인 파일 web_lib.p1에 배치하시오. 이 루틴들은 CGI프로그람에 의하여 실행시에 요구될것이다. 언제나 참의 값을 귀환하도록 파일의 끝에 명령문 1;을 추가하여야 한다.

20.21.2 CGI주프로그람 emp_add.pl

세번째 부분루틴을 취급하기전에 URL에서 지적된 CGI프로그람 emp_add.p1을 보기로 하자. 이것은 본문자료기지에 한행을 추가하기 위하여 방금 론의된 두개의 부분루틴을 호출한다.

```
$ cat emp_add.pl
#!/usr/bin/perl
require "web_lib.pl" ;

open (OUTFILE, ">>/home/sumit/public_html/emp_out.lst");
&Parse(*field) ;
print "Content-type: text/html\n\n";
$Htmlheader("Testing Query String", "The QUERY_STRING Variable");
```

```
print "The query string is $ENV{'QUERY_STRING'}<br>\n";
print "The method of sending data to server is$ENV{'REQUEST_METHOD'}<br>\n";
print "THE content length is $ENV{'CONTENT_LENGTH"}<br>\n";
print OUTFILE "$field{'empid'}|$field{field{'ename'}|$field{'desig'}|$field{dep t'}|
$field{'dtbirth'}|$field{'salary'}<br>\n";
print "A record has been aded <a href="\http://locallhost/cgi-bin/emp_query.pl\"
>Click here to see the records</a><br>\n" ;
&Htmlfooter ;
close (OUTFILE);
```

이 프로그람은 HTML을 만들어야 하므로 그의 내용형의 맞춤법(spell)을 명백히 지켜야 하며 자료를 반대로 보내기전에 빈 행을(\n\n) 두어야 한다. HTML의 머리부를 Htmlheader부분루틴으로서 출력되며 꼬리부는 Htmlfooter에 의하여 끝부분에 출력된다.

open명령문으로부터 스크립트는 파일 emp_out.lst에 쓰기를 진행한다. 봉사기에로 양식자료를 전송하는 HTTP새끼프로쎄스는 보통의 사용자(24.14)처럼 실행된다. 이 프로쎄스에서 파일을 만들수 있게 하자면 등록부 public_html은 객관쓰기가능(world-writable)으로(chmod 777로써) 되여야 한다. 이것은 처음에 내용을 기입하기 위하여 필요하다. 일단 파일이 만들어 지면 등록부는 자기의 원래의 허가권을 가질수 있다.

20.21.3 Parse부분루틴

Parse는 여기서 사용되는 세번째 부분루틴이다. Parse는 매개의 <이름=값>형식의 쌍을 조합배렬 %field에서 개별적인 항목으로서 유효하게 만든다. 배렬 %field는 emp_add.p1스크립트내에서 *로서 Parse에 참조형으로 넘겨 진다.

&Htmlheader다음의 세개의 print명령문은 봉사기의 환경변수들의 내용을 열람기창문에 표시한다. 네번째 print명령은 자료기지에 한 행을 추가하기 위하여 파일조종자 OUTFILE을 사용하며 구분문자로서 |기호를 사용한다. 마지막 print명령문은 완료통보문을 현시하고 A HREF를 리용하여 emp_query.p1프로그람에로 하이퍼련결을 제공한다. 이 련결을 찰칵하면 파일 emp_out.lst의 모든 행(방금 추가한것까지도 포함하여)들을 다 볼수 있어야 한다.

perl이 조합배렬 %field에서 양식값들을 어떻게 유효하게 만드는가를 리해하자면 Parse부분루틴에 대하여 명백히 연구하여야 한다.

```
sub Parse {
   local (*in) = @_ ;
   local ($i, $key, $val) ;              # 국부변수

   if ($ENV{'REQUEST_METHOD'} eq"GET") {
      $in=$ENV{'QUERY_STRING'} ;
   } elsif ($ENV{'REQUEST_METHOD'} eq "POST") {
      read(STDIN,$in, $ENV{'CONTENT_LENGTH'}) ;
```

```
    }                # $in에 질문문자렬을
    @in = split(/&/,$in);              # <이름=값>의 쌍으로 분할한다
    foreach $i (0 .. $#in) {
    $in[$i] =~ s/\+/ /g ;              #  +를 공백으로 복호화한다
    ($key, $val) = split(/=/,$in[$i], 2) ;            # 먼저 =를 기준으로 하여 분할한다
    $key =~ s/%(..)/pack("c",hex($1))/ge;
    $val =~ s/%(..)/pack("C",hex($1))/ge;
    $in{$key} = $val ;              # 조합배렬에서의 이름과 값
    }
    return %in;
}
```

여기서 Parse는 참조로서 배렬을 받았다. 이 배렬은 부분루틴안에 있는 %in에 복사된다. 방금 론의한 봉사기의 세개의 환경변수를 평가하는데 조합배렬 %ENV를 사용하였다. 질문문자렬은 사용되는 메써드에 관계없이 변수 $in에 할당된다. POST된 자료는 표준입력으로부터 read함수에 의해 $in으로 읽혀 진다. 읽어 들일 문자의 개수는 read의 세번째 인수(내용길이)에 의하여 결정된다.

질문문자렬이 <이름=값>형식의 쌍들에 대한 구분문자로서 &를 사용하므로 첫번째 split는 이러한 쌍 모두를 스칼라배렬 @in에 기억시킨다. 자료에 공백이 있을 때마다 그것을 +로 부호화하였으므로 s함수는 다시 +를 공백으로 복호화한다. 많은 문자들은 URL문자렬에서 특별한 의미를 가지고 있으므로 16진문자렬로 부호화된다. 실례로 자료마당의 요소들을 분리하는 /기호는 URL문자렬에서는 등록부들을 경계 지어 준다. 그러므로 그림 20-3에서 볼수 있는바와 같이 질문문자렬이 봉사기에로 보내지기전에 %2F로 부호화된다.

그림 20-3. CGI프로그람 emp_add.pl의 출력

pack함수는 이 16진값들을 본래의 ASCII문자들로 변환한다. s함수는 이러한 문자들을 TRE를 사용한 패턴 %(..)으로서 식별한다. ge기발들은 pack가 문자그대로서 취급하지 않고 표현식으로서 해석하도록 한다.

foreach순환은 배렬 @in으로부터 매개의 <이름=값>형식의 쌍을 선택한다. 복호화후 배렬의 매개 요소는 다시 =기호에서 분리되며 변수 $key와 $val들에 보존된다. 첫번째는 첨자로서 그리고 다음의것은

조합배렬 %in에 값으로서 설정된다. 많은 기호들은 URL문자렬에서 특별한 의미를 가지고 있으므로 16진 문자렬로 부호화된다. 이 배렬은 호출한 프로그람에로 돌려 진다. 이 프로그람에서는 in을 변수 $in으로서,또 스칼라목록 @in으로서, 조합배렬 %in으로서 아무러한 장애도 없이 사용하고 있다.

20.21.4 질문프로그람 emp_query.pl

그림 20-3은 양식의 Add단추가 눌리워 진후에(그림 20-1) 열람기창문에 나타나는 emp_add.pl의 출력이다. 방금 추가한것도 포함하여 자료기지의 모든 행을 보기 위하여 제공되는 하이퍼련결을 주시하시오. 이 하이퍼련결을 누르면 emp_qnery.pl프로그람이 실행되며 매 사람들의 목록을 표의 요소로 하여(그림 20-4) 표시된다. 아래에 그 프로그람을 보여 준다.

```
$ cat emp_query.pl
#!/usr/bin/perl
require "web_lib.pl" ;

open (OUTFILE,"/home/suit/public_html/emp_out.lst");
print "Content-type: text/html\n\n" ;
&Htmlheader("Retrieving from Database","Result of Query:");
print "<table border=1 bordercolor=magenta bgcolor=cyan>";
print "<tr><th>Emp-id</th><th>Full Name</th><th>Designation</th>" ;
print "<th>Department</th><th>Date of Birth</th><th>Salary (\$)</th></tr>";
while (<OUTFILE>) {
    ($empid,$ename,$desig,$dept,$dtbirth,$salary) = split (/\|/) ;
    print "<tr><td>$empid</th><td>$ename</td><td>$desig</td>" ;
    print "<td>$dept</td><td>$dtbirth</td><td>$salary</td></tr>" ;
}
print "</table>" ;
&Htmlfooter ;
```

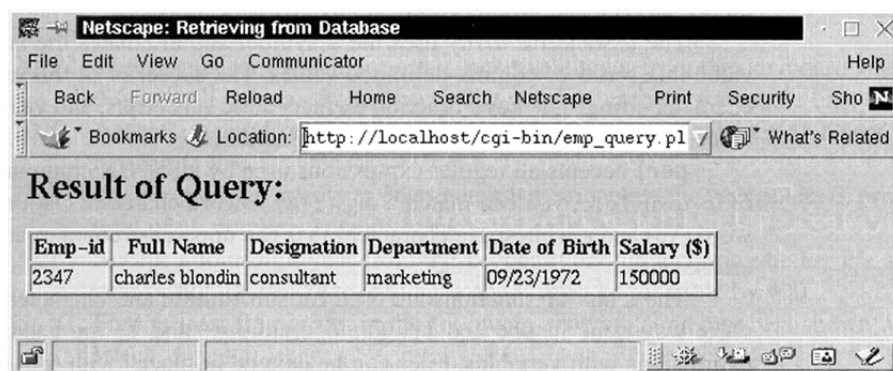



그림 20-4. CGI프로그람 emp_query.pl의 출력

여기서 파일 emp_out.lst는 읽기방식으로 열린다. 표의 머리부는 <tr>와 <th>꼬리표들로서 출력된다. 프로그람은 OUTFILE의 매행을 취하여 개별적인 요소로 분할하고 <tr>와 <td>꼬리표를 가지고 표의 렬로서 그것을 표시한다. perl은 CGI가 아니라 다른 본문처리에 대해서도 다른 프로그람언어를 쓰고 싶지 않을 정도로 아주 쉽고 간단하다. perl이 인터네트에서 CGI프로그람작성에 가장 널리 사용되는 언어로 되는것은 당연한 일이다.

주해 CGI는 봉사기관리자가 흔히 개별적인 사용자들에 의한 CGI조작을 불가능으로 만들기때문에 인터네트에서 보안이 철저하다. 자기 체계에 이러한 제한이 있다는것을 알고 있는이상 관리자와 접촉교제하시오.

요 약

perl은 grep, tr, sed, awk와 쉘의 상위모임이다. perl프로그람은 -e선택항목으로서 지령행에서 리용될수 있다. 그러나 흔히는 해석기(파일의 첫행에 놓인다.)와 함께 파일에 놓여 진다. perl의 모든 명령문들은 반두점으로 끝난다.

입력은 파일조종자 <STDIN>을 할당하는것으로써 건반으로부터 변수에 읽혀 진다. perl은 공백과 행바꾸기문자를 포함하여 입력된 모든것을 다 읽는다. 행의 마지막문자는 chop함수에 의해서 제거된다.

변수들은 대입과 평가에 대하여 둘 다 기호 $를 요구한다. 변수들에 대해서는 형정의가 필요 없으며 아주 높은 정확도를 가진 수값도 보관할수 있다. 변수의 값에는 행바꾸기(\n) 혹은 타브(\t)와 같은 확장문자렬(escape sequences)들도 포함될수 있다. 문자렬의 첫 문자는 \u로써, 전체 문자렬은 \U로써 대문자로 변환할수 있다.

점(.)은 문자렬련결에 사용되며 x는 반복에 리용된다. substr는 문자렬의 오른쪽과 왼쪽 즉 량쪽으로부터 부분문자렬을 추출하며 또한 어떤 문자렬을 삽입할수도 있다. uc와 ucfirst는 각각 전체의 인수와 첫 문자를 대문자로 변환한다.

perl에서는 -n선택항목을 리용하여 파일을 읽기 위한 무조건순환을 설정할수 있다. 순환은 지령행에 지적된 파일이름을 가리키는 기호 <>을 리용하여 설정될수 있다. while(<>)은 스크립트에서 파일을 읽기 위하여 흔히 사용된다.

$.은 현재의 행번호를 기억하며 범위연산자(..)는 행묶음을 지적한다.

$_은 많은 perl함수들에서 리용되는 기정변수이다. 이것은 읽혀 진 마지막행 혹은 정합된 마지막패턴을 기억하고 있다. print, chop, split, 패턴정합과 치환은 기정적으로 $_에 대하여 조작한다.

perl은 목록과 배렬을 광범하게 사용한다. @arr는 변수에 대입될 때에는 배렬의 길이를 표현한다. $#arr는 배렬 @arr의 마지막첨수를 기억하고 있다. @ARGU[]는 모든 지령행인수들을 보관하고 있으며 지령이름은 $0으로서 유효하다.

foreach순환은 배렬을 리용하며 그의 매 요소들은 변수에 차례로 대입된다. @_은 기정배렬이며 공백은 기정경계구분문자이다. split는 목록을 변수나 배렬로 가른다. split행의 요소들은 join함수를 리용하여 서로 결합될수 있다. 요소들은 배렬에서 삭제(shift와 pop)되거나 삽입(unshift와 push)될수 있다. splice는 임의의 배렬위치에 대하여 모든것을 다할수 있다.

조합배렬은 %기호를 사용하며 배렬첨자와 그에 해당한 값이 반점으로 구분되면서 엇바뀌여 놓인다. 조합배렬의 첨자는 문자렬일수도 있다. keys함수는 이러한 첨자들을 추출하며 values는 값들을 려과한

다. sort는 추출결과를 순서로 정돈하기 위하여 사용된다.

perl에서는 모든 UNIX지령들에서 사용되는 정규식들을 다 리용할수 있으며 자기자체의 고유한 정규식들을 가지고 있다. 10진수는 \ d, 단어문자는 \ w, 단어의 시작부분은 \ b, 공백(whitespace)은 \ s으로 정합시킬수 있다. 대문자로 되여 있는것들은 소문자로 되여 있는 경우와 반대의 기능을 수행한다.

s와 tr함수들은 sed와 tr에서 했던것과 같은 방법으로 치환과 문자변환에 사용된다. 연산자들인 =~와 !~는 변수들과 정규식을 정합하는데 사용된다. 파일들은 -i선택항목을 리용하여 그자리에서 편집될수 있으며 개별적인 확장자를 가지고 여벌복사할수 있다.

IRE와 TRE는 \이 ()와 {}기호들앞에 쓰이지 않는다는것을 제외하고는 이전과 같은 방법으로 동작한다. 묶음화된 패턴은 다음번 묶음화가 진행될 때까지 $1, $2, ...으로 임의의 곳에서 재생될수 있다.

perl은 파일을 호출하는데서 파일조종자를 사용한다. 파일조종자는 관흐름을 표현할수도 있다. print는 파일에 쓰기를 진행하기 위하여 파일조종자를 리용한다. select파일조종자명령문은 print가 기정으로 파일조종자를 사용하게 한다.

perl의 파일검사는 파일의 《나이》(변경과 호출)를 소수점아래의 여러자리까지 기억할수 있다.

부분루틴들은 &를 리용하여 호출되며 그의 인수들은 배렬 @_에 기억된다. 부분루틴들은 외부파일에 보관되며 파일의 끝에 명령문 1;이 반드시 있어야 한다. 호출하는 프로그람은 require명령으로서 외부파일에 놓여 있는 부분루틴들을 포함한다.

perl은 CGI프로그람작성을 위한 언어이다. perl프로그람은 <form>끄리표의 action속성에 지적될수 있다. GET메쏘드는 자료를 QUERY_STRING환경변수를 통하여 프로그람에 보낸다. POST자료는 표준입력으로서 보내진다. 프로그람은 <이름=값>형식의 쌍들을 추출하며 이름과 값을 분리하여 열람기에 보내지게 될 HTML코드를 만든다.

시험문제

1. 다음의 프로그람에서 무엇이 틀렸는가?. 어떤 방법으로 출력된다고 생각되는가

```
#!/usr/bin/perl
x = 2;
print x ** 32;
```

2. 파일의 모든 행들에 행번호를 붙이시오. 이때 행번호와 행사이에 타브만큼의 공백을 두시오.
3. GUID로서 100을 가지고 있는 /etc/passwd의 행들을 추출하시오.
4. 순환을 리용하지 말고 문자렬 UNIX를 20번 출력하시오.
5. 어떤 파일의 모든 문자들을 쉘의 방향절환을 사용하지 말고 대문자로 어떻게 변환시킬수 있는가.
6. 건반으로부터 정의 옹근수를 받아 들여 1부터 그 수까지의 모든 옹근수들을 크기순서로 각각 개별적인 행에 표시하는 프로그람을 작성하시오.
7. 건반으로부터 문자렬을 받아 들여 문자렬의 매개 문자를 개별적인 행에 출력하시오.
8. 사용자에게 수값입력을 반복적으로 요구하는 프로그람을 작성하시오. 만일 사용자가 0을 입력하면 지금까지 입력한 모든 수들을 표시하시오.
9. 건반에서 네자리10진수를 받아 들여 그것이 윤년인지 아닌지 검사하시오(풀이방향: 00으로 끝나는

해는 400으로도 나누어 져야 한다).

10. 다음의 프로그람에서 적어도 네개의 오유를 찾으시오(행번호는 왼쪽에 보여 준다).

```
1   #/usr/bin/perl
2   print "What is your age ?;
3   $a = <STDIN>
4   chop ($a);
5   if ( $a < 18 )
6        print "Not old enough to vote yet /n";
7    } else {
8        print "You are old enough to vote" ;
9    }
```

11. 문자렬과 수값을 입력할 프롬프트를 제시하고 들어 온 문자렬을 그만한 회수만큼 문자렬이 개별적인 행에 표시되도록 하는 프로그람을 작성하시오.

련습문제

1. perl로 파일을 어떻게 두줄공간화하겠는가?
2. 행의 시작부분에 행번호가 아니라 문자 A, B, C,등으로 추가하시오.
3. 행들을 어떻게 반대순서로 표시할수 있겠는가.?
4. 파일에서 처음으로 나타나는 문자렬을 어떻에 출력할수 있는가(문자렬과 파일이름은 각각 첫번째와 두번째 인수이다)?
5. 파일에서 모든 단어의 첫 문자를 대문자로 표시하시오.
6. 2진수(인수로서 제공됨)를 10진수로 어떻게 변환할수 있겠는가?
7. 세개의 련속적이며 동일한 자모문자로 되여 있는 문자렬들(aaa 혹은 bbb와 같이)을 찾아 내시오.
8. 하나이상의 파일들에서 사용되는 모든 단어들을 목록으로 표시하고 그의 계수값을 현시하는 스크립트를 작성하시오(형식->단어:계수값).
9. <변수=값>형식으로 모든 환경변수들의 목록을 어떻게 표시할수 있겠는가?
10. find와 perl을 리용하여 한해전에 변경된 모든 파일들을 삭제하시오. find를 -exec rm과 함께 리용하는 방법과 비교해 볼 때 무엇이 개선되였는가?
11. 현재등록부의 모든 perl스크립트들에서 해석기행을 #!/usr/local/bin/perl로 변경시키시오.
12. HTML의 닫기꼬리표들은 /로 시작한다. 실례로 는 로서 닫긴다(속성이 없이). 하나의 단어를 포함하고 있는 이러한 모든 꼬리표들을 대문자로 바꾸어 보시오(<img src=....은 변경되지 않을 것이다).
13. 현재등록부에서 다중련결을 가지고 있는 파일들의 목록만을 표시하시오.
14. 15.12.2의 HTML문서들에서 URL들을 변경시키는 실례를 참고하여 그것을 perl로 실현해 보시오. 여기서 URL들을 참조하는 <IMG SRC>에 주의를 돌리시오.
15. 자기의 열람기에 df지령출력을 표시하는 CGI스크립트를 작성하여 보시오.s

제 21 장. 체계관리자의 립장에서 본 파일체계

지금까지 우리는 UNIX파일체계를 하나의 거대한 나무모양의 구조체로 보아 왔다. 그러나 그렇게 되는것은 드물다. 사실상 그것은 흔히 그러한 구조체 즉 파일체계들의 어떤 결합이다. 이러한 파일체계들은 대체로 UNIX체계의 가장 섬세한 구성요소들이다. 이 장에서 우리는 파일체계를 자세히 따져 보며 그 내부의 몇가지를 리해하게 될것이다. 우리는 구획과 파일체계들을 만들고 그것들을 태우며(mount) 검사하는것을 배우게 될것이다.

파일체계는 파일과 등록부, 장치들과 관계되는 모든 정보를 관리한다. 그것을 안전하고 정확한 상태로 관리하는것이 체계관리자의 일감이므로 그는 그 내부를 잘 리해하여야 한다. 즉 때때로 제기될 우려가 있는 모순점들을 고칠수 있어야 하며 자료류실이 최소로 되도록 하여야 한다. 체계관리자는 체계가 기동되지 않을 때 당황하여 체계전반을 다시 설치하여서는 안된다. 그가 거의 모든 손상을 바로 잡을수 없다면 그것은 엄중한 결함이라고 보아야 할것이다. UNIX체계가 파일체계를 관리하는데 필요한 도구들을 제공하므로 이 임무를 두려워 할 필요는 없다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 장치파일들의 두가지 형태와 그 이름들의 의미를 배운다(21.1, 21.2).
- 파일들이 개별적인 구획들과 파일체계들에 어떻게 보관되는가를 배운다(21.4).
- 파일체계의 4가지 구성요소들의 기능을 배운다(21.5).
- 색인마디가 파일이 사용하는 모든 디스크블로크주소들의 경로를 어떻게 보관하는가를 배운다(21.5.3).
- 핵심부의 파일처리에서 등록부의 역할을 파악한다(21.6).
- UNIX체계에서 보게 될 여러가지 형태의 파일체계들을 배운다(21.8).
- Linux기계상에서 fdisk와 mkfs를 리용하여 구획과 파일체계를 만드는 방법을 배운다(21.9).
- mount와 umount지령을 리용하여 파일체계를 태우고 내리우는 방법을 배운다(21.10).
- mount가 /etc/fstab를 리용하여 어떻게 태우기정보를 얻는가를 파악한다(21.10.3).
- fsck를 리용하여 파일체계를 검사하고 복구하는 방법을 배운다(21.11).

주해

체계관리지령들은 체계에 강하게 의존하며 그것들의 기능과 출력은 체계에 따라 크게 달라진다. 때때로 어떤 체계에서 지령이 전혀 유용하지 않을수도 있다. 이 장에서 언급된 지령을 실행시킬수 없는 경우에는 체계문서를 보는것이 좋다. 지령이 다른 이름을 가지거나 다른 선택항목을 사용할수 있다.

이 장에서 지적한 거의 모든 지령들을 사용하기 위하여서는 뿌리사용자로서 가입하여야 한다. 필요하다면 상급사용자권한을 획득하는 동작을 잘 알기 위하여 22.2를 보시오. 이 장에서는 대체로 뿌리사용자가 리용하는 프롬프트 #에서 작업하게 된다.

21.1 장치

모든 장치들도 다 파일이다. 파일에서와 같이 장치도 열고 읽거나 쓴 다음 닫는다. 이 모든것을 수행하는 기능은 핵심부안에 체계의 모든 장치별로 내장되여 있다. 모든 장치파일들은 /dev나 그의 보조등록부들에 보관된 System V를 실행시키는 체계에서 이 장치들의 간단한 목록을 아래에서 보여 준다.

```
$ ls -l /dev
total 52
brw-rw-rw-   1  root     sys        51,    0  Aug  31  07:28  cd0      CDROM
brw-rw-rw-   2  bin      bin         2,   64  Feb  23   1997  fd0      기정플로피구동기
brw------    1  sysinfo  sysinfo     1,    0  May   7   1996  hd00     첫 하드디스크
crw------    2  bin      bin         6,    0  Dec   5  14:12  lp0      인쇄기
cr--r--r--   1  root     root       50,    0  Aug  31  07:28  rcdt0    테프구동기
crw------    1  henry    terminal    0,    0  Oct  15  10:23  tty01    말단
crw-rw-rw-   2  bin      bin         5,    0  May   7   1996  tty1a    직렬포구 1
crw-rw-rw-   1  bin      bin         5,  128  Feb  23   1997  tty1A    모뎀포구 1
```

SVR4는 몇가지 파일들을 더 포함한 두개의 추가적인 등록부 /dev/dsk와 /dev/dsk도 가지고 있다. 이 등록부안의 파일들은 때때로 /dev에 동일한 파일(또는 련결)을 가지고 있다. 실지로 이러한 목록들은 우에서 보여 주는것보다 대단히 크며 콤퓨터의 주기억까지도 포함하여 체계의 모든 가능한 장치를 포함한다. 이 목록은 두가지 중요한 점을 보여 준다.

- 장치파일들은 허가권마당의 첫 문자(b 또는 c)에 따라서 기본적으로 2개의 부류로 갈라 진다.
- 일반적으로 다른 파일들에서 크기를 표현하는 다섯번째 마당이 한쌍의 수들로 이루어 진다. 장치파일은 자료를 포함하지 않는다.

이 속성들의 의미는 다음에 취급된다.

21.1.1 블로크장치와 문자장치

먼저 디스크읽기와 쓰기에 대하여 보자. 파일을 보관하는 명령을 줄 때 그 요구는 다른 사용자들의 요구와 결합되며 쓰기조작이 덩어리(chunk)나 블로크단위로 진행된다. 여기서 매 블로크는 몇개의 디스크분구들을 표현한다. 디스크로부터 읽을 때 에는 가장 최근에 사용된 자료를 포함하고 있는 **고속완충기억기**(buffer cache)가 먼저 접근된다. 거기서 자료가 발견되면 디스크접근은 취소된다. 이것은 많은 시간을 절약한다. 이 기능을 무시하고 장치를 직접 접근하도록 할수도 있다. 대부분의 장치들은 사용자로하여금 그렇게 할것을 허용하며 접근방법은 호출되는 장치의 이름에 의하여 결정된다.

허가권마당에서 첫 문자는 일반적으로 b 또는 c이다. 플로피구동기, CD-ROM, 하드디스크들은 허가권이 b로 시작된다. 이 장치들에서 모든 자료는 블로크단위로 읽기, 쓰기되며 고속완충기를 사용한다. 그것은 이 장치들이 **블로크형장치**(block special device)로서 참조되기때문이다. 다른 한편 말단과 테프구동기, 인쇄기들은 **문자형장치**(character special device) 또는 **미가공장치**(raw device)들이다(문자 c가 그것을 가리킨다). 읽기, 쓰기조작은 고속완충기를 무시하고 직접에 그 장치를 접근한다.

주해

System V에서는 많은 장치들이 문자형과 블로크형을 다 가진다. 하드디스크, 플로피구동기, CD-ROM들은 블로크장치와 문자장치를 다 가지고 있다. 일반적으로 블로크장치의 이름앞에 r가 붙으면 그 장치는 문자장치로 된다. 또한 블로크장치들은 /dev/dsk에서, 문자형장치들은 /dev/rdsk에서 찾을수 있다.

21.1.2 기본번호와 보조번호

어떤 장치를 동작시키는데 필요한 루틴들의 모임을 **장치구동프로그람**(device driver)이라고 한다. 어떤 장치가 접근될 때 핵심부는 그것이 옳게 동작하도록 정확한 장치구동프로그람을 호출하고 몇개의 파라메터들을 보낸다. 핵심부는 장치의 형뿐아니라 플로피의 밀도나 디스크의 구획번호와 같은 그 장치에 대한 일정한 세부정보들도 알아야 한다.

앞에서 본 목록의 다섯번째 마당은 바이트단위의 파일크기가 아니라 반점으로 구분되는 두개의 수를 보여 준다. 이 수자들을 각각 **기본장치번호**(major device number)와 **보조장치번호**(minor device number)라고 부른다. 기본번호는 장치구동프로그람을 표현하며 사실상 장치의 형이다. 같은 조종기에 결합되여 있는 모든 하드디스크들은 같은 기본번호를 가질것이다.

보조번호는 핵심부가 장치구동프로그람에 보내는 파라메터들을 나타낸다. 흔히 이것은 장치의 특수한 특성들을 표시한다. 실례로 fd0h1440과 fd1h1440은 어떤 조종기에 결합된 2개의 플로피장치를 표현한다. 그러므로 이 두 장치는 동일한 기본번호와 서로 다른 보조번호를 가질것이다.

장치파일들은 또한 동일한 의미의 허가권을 가진다. 말단에 출력를 보내기 위해서는 그 장치에 대한 쓰기허가권을 가지는것이 필요하며 플로피디스크를 읽기 위해서는 그 장치파일에 대한 읽기허가권을 가지고 있어야 한다. 그러나 대부분의 장치허가와 기타 속성들은 체계관리자만이 설정할수 있다.

21.2 장치이름의 의미

UNIX장치파일들의 고유한 특징은 동일한 장치가 서로 다른 몇개의 파일이름으로 접근될수 있다는것이다. 이것은 때때로 역방향호환성(backward compatibility)을 위해서, 개별적인 장치에 특별한 기능을 부여하기 위해서 수행된다.

UNIX와 Linux체계들에서 보통 부닥치게 되는 장치들을 표 21-1에서 보여 준다.

표 21-1. 대표적인 장치이름들(등록부: /dev)

SVR4장치	Linux장치	의미
Cd0 혹은 dsk/c0t6d0s2	cdrom	CD-ROM
fd0 혹은 diskette	fd0	기정플로피구동기
dsk/f0q18dt	fd0H1440	1.44MB 플로피
rdsk/f0q18dt	fd0H1440	1.44MB 미가공플로피
hd00 혹은 dsk/c0t0d0s2	hda	첫번째 하드디스크
hd10 혹은 dsk/c1t3d0s2	hdb	두번째 하드디스크
lp0	lp0	인쇄기
rcdt0 혹은 rmt/0	st0	테프구동기
term/1	tty1	말단
tty1a	cua0	직렬포구 1
tty2A	ttyS1	모뎀포구 2

Linux에서는 장치이름들이 아주 불변적이지만 System V에서는 그렇지 않다. 서로 다른 SVR4체계들은 흔히 서로 다른 이름들을 가진다. 만일 Solaris나 HP-UX를 사용하고 있다면 그 이름들이 다를것이다.

장치이름들에서는 많은 0과 1 그리고 다른 수들을 보게 될것이다. 흔히 파일이름으로 장치를 식별할수 있다. System V장치파일들은 /dev/dsk(문자장치들은 /dev/rdsk)에 상주한다. 이름 /dev/dsk/f0q18dt는 3.5인치플로피장치(블로크형)를 표현한다. 이 장치는 기동가능(0)하며 4배밀도이고(q) 자리길당 18개의 분구를 가지는 1.44MB디스케트이다. 많은 장치들에는 기정이름도 있다. 즉 플로피구동기는 /dev/fd0으로 접근될수 있다.

이전의 체계들에서는 3.5인치플로피구동기를 표현하는 파일 fd0135ds18을 볼수도 있다. 그 장치는 기동가능(0)하고 량면이며 135개의 자리길과 자리길당 18개의 분구를 가진다. 이 장치는 문자형장치 /dev/rfd0135ds18로서도 호출될수 있다. 이 파일들은 XENIX체계들에서 사용되며 역방향호환성을 가지고 제공되고 있다.

주해

일반파일들이나 등록부파일들과는 달리 장치파일들은 어떤 자료도 포함하지 않는다. 그것들은 단순히 실제적인 물리적장치를 지정한다

Linux

장치이름과 의미

Linux파일들은 대체로 /dev안에 존재한다. 그것들의 이름도 완전히 다르다.

```
lrwxrwxrwx   1  root   root       3  Mar   11  20:54  cdrom -> hdc
crw-rw----   1  root   uucp   5,  64  Nov   30  18:55  cua0                직렬포구 1
brw-rw-rw-   1  root   root   2,  40  Nov   30  18:55  fd0H1440            첫 플로피
brw-rw-rw-   1  root   root   2,  41  May    1  04:19  fd1H1440          둘째 플로피
brw-------   1  root   root   3,   0  Nov   30  18:55  hda               첫 하드디스크
crw-rw----   1  root   lp     6,   0  Nov   30  18:55  lp0                     인쇄기
lrwxrwxrwx   1  root   uucp      10  Apr   24  15:04  modem -> /dev/ttyS0
lrwxrwxrwx   1  root   root       9  Apr   24  15:04  mouse -> /dev/cua1
crw-------   1  henry  tty    4,   1  Apr    3  17:31  tty1                       말단
crw-rw----   1  root   uucp   4,  64  Nov   30  18:55  ttyS0               직렬포구 1
```

Linux는 3.5인치 기동가능한 플로피구동기를 위하여 블로크장치 /dev/fd0H1440만을 사용한다. 그것은 또한 장치파일의 련결체계도 광범히 지원한다. /dev/modem은 /dev/ttyS0에 련결되여 있다(첫 송신직렬포구). CD-ROM은 하드디스크를 표현하는 장치들중 하나에 련결된다(/dev/hdc).

21.3 하드디스크

체계관리자는 구획과 파일체계를 생성할 때 하드디스크용어를 알아야 한다. 매 디스크는 한개 혹은 여러개의 **원판**(platter)을 가지며 매 원판은 2개의 표면을 가진다. 자두는 매 표면을 읽거나 쓴다. 만일 8개의 유용한 표면이 있다면 자두가 필요하다. 자두들은 나란히 움직이며 그것들의 운동은 개별적으로 조종될수 없다.

매 표면은 련번호가 붙은 여러개의 동심원모양의 **자리길**(track)로 구성된다. 같은 자리길번호가 붙은 자리길이 표면개수만큼 있게 된다. 그러므로 매 디스크표면상의 같은 번호를 가진 모든 자리길들이

하나의 **원통**(cylinder)을 이루는것을 볼수 있다. 디스크안에는 매 유용한 표면상의 자리길개수만큼의 원통이 있다. 매 자리길은 더 나아가서 **분구**(sector) 혹은 **블로크**(block)로 쪼개진다. 그러므로 만일 매 자리길이 32개의 분구를 가지며 디스크 한개가 8개의 표면을 가진다면 매 원통에 256개의 분구가 있게 된다(그림21-1).

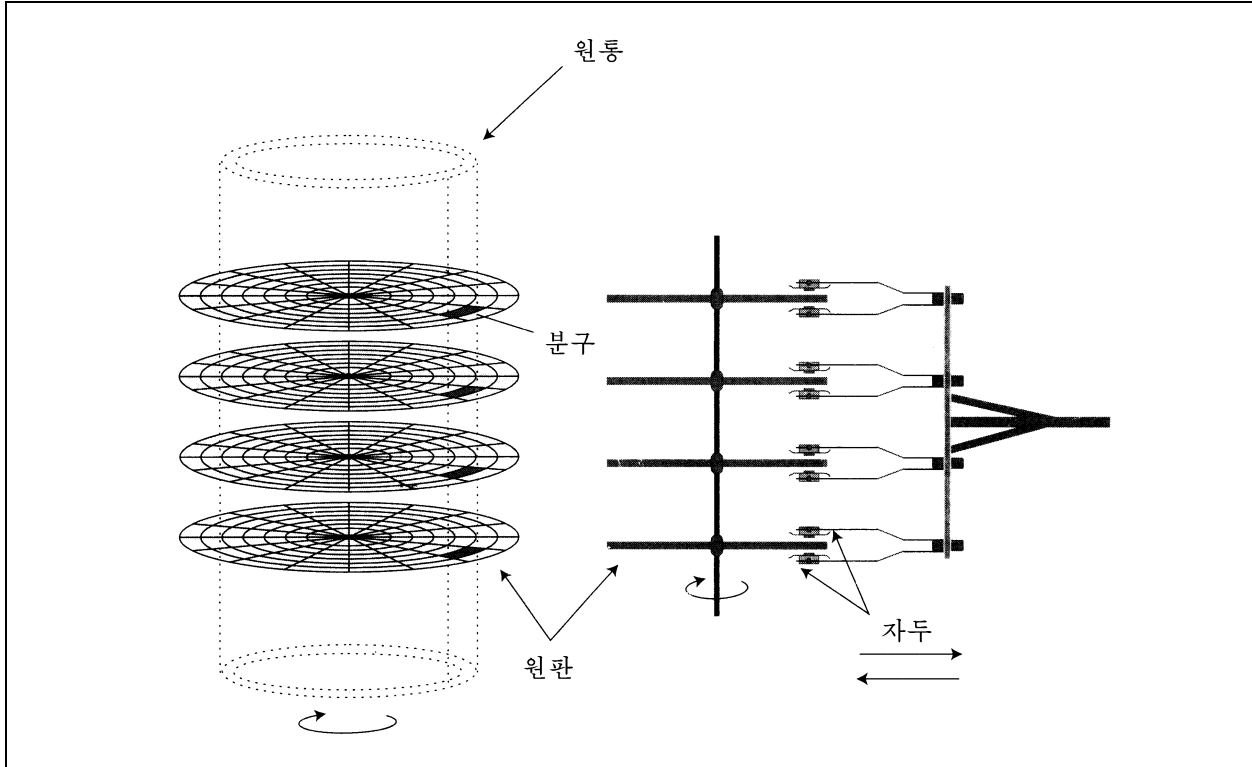



그림 21-1. 하드디스크

디스크는 끊임없이 회전한다(보통 3600rpm). 디스크자두들은 자리길들사이를 반경방향으로 움직이며 자두가 부분적인 자리길상에 위치하였을 때 그 자리길의 모든 분구들이 매우 짧은 시간동안에 그 자두를 통과한다.

21.4 구획과 파일체계

다른 조작체계들과 마찬가지로 UNIX는 초기화된 하드디스크를 요구한다. 초기화조작은 조작체계령역의 밖에 놓인다. 초기화조작은 불량한 자리길들을 표식하여 읽기쓰기조작이 그것들을 피할수 있게 한다. 이 조작은 특수한 도구에 의하여 수행되군 하였으나 오늘날 디스크들(IDE와 SCSI)은 미리 형식화되여 나오므로 근심할 필요가 없다.

디스크를 사용하려면 그것을 **구획**(partition)들로 나누어야 한다. 매 구획은 론리적으로 독립적인 디스크처럼 간주되며 자체의 장치파일에 의해 접근된다. 구획은 하나 혹은 그이상의 **파일체계**(file system)를 만들지 않는한 그자체만으로는 사용할 준비가 되는것이 아니다. 구획과 달리 파일체계는 UNIX체계의 여러가지 구성요소들을 포함하는 등록부구조를 가진다. 마지막으로 작업이 쉬워 지도록 하기 위해 모든 파일체계들이 하나의 파일체계를 형성하도록 결합되여야 한다.

21.4.1 구획

구획에 대하여 먼저 보기로 하자. 이 용어는 BSD UNIX와 Solaris의 **쪼각**(slices)과 호환성이 있다. 디스크를 여러개의 분리된 구획들로 나누는것은 관리의 면에서 우점을 가진다.

- 분리된 구획들은 여러개의 자료구역들사이에 일어 날수 있는 가능한 충돌을 막는다. 사용자자료가 끊임없이 그리고 예측할수 없게 불어 날 때 체계의 기본구획에로 넘쳐 나는것은 허용되지 말아야 한다.
- 만일 한 구역에 불량이 있으면 다른 구역들은 이 나쁜 영향으로부터 효과적으로 보호된다. 체계관리자는 체계를 닫지 않고 어떤 구획을 수리할수 있다.
- 작은 크기로 많이 분할하면 쪼각화가 제한될것이다. 쪼각화는 디스크능률에 불리한 영향을 미친다.
- 만일 체계가 적당한 수의 구획을 가지면 매 구획은 각각 한개의 테프에 따로따로 여벌복사될수 있다. 관리자는 여러 구획들을 위하여 서로 다른 여벌복사일정을 가질수도 있다.
- 2중기동체계(Linux와 SCO UNIX와 같은)들은 서로 다른 조작체계들을 위해 분리된 구획들을 요구한다.

구획을 만들 때에는 구획이 항상 원통경계에서 시작되고 끝나는가를 확인해야 한다. 만일 어떤 구획이 어느 한 원통의 중간에서 시작된다면 디스크능률이 낮아 진다. 구획을 만들기 위한 편의프로그람들은 보통 이것을 가장 가까운 완전한 원통에로 맞추어 주지만 그렇지 않은 경우에는 주의해야 한다. 원통대신에 자리길을 지정하도록 요구하는 경우에 계산을 정확히 하지 않으면 경계를 놓치기 쉽다.

주해

2중기동체계(Linux와 SCO UNIX와 같은)들은 여러개의 조작체계들이 같은 기계상에서 기동될수 있게 한다. 그 경우에 매 조작체계는 각각의 분리된 구획을 요구한다. 체계는 기동될 조작체계를 사용자에게 문의한다.

21.4.2 파일체계

디스크가 분할된후에 매 구획에 **파일체계**(file system)를 만들어야 한다. UNIX체계는 보통 여러개의 파일체계로 구성되며 매 파일체계는 뿌리(root)를 선두로 하는 자체의 등록부나무를 가진다. 기계상에 여러개의 뿌리가 보이지 않는것은 모든 파일체계들이 사용시에 하나로 되기때문이다. 이 모든것이 어떻게 일어 나는가는 후에 설명되겠지만 여기서는 먼저 파일체계가 어떻게 조직되는가를 보기로 한다.

매 파일체계는 1024byte의 순차적인 블로크들로 조작되며 보통 다음과 같은 4개의 구성요소들로 되여 있다.

- 기동블로크(boot block): 이 블로크는 작은 기동프로그람과 구획표를 포함한다.
- 상위블로크(superblock): 파일체계에 대한 전체적인 정보를 포함한다. 추가적으로 파일생성시 핵심부에 의하여 즉시적으로 할당될수 있는 색인마디(26.5)들과 자료블로크들의 자유목록도 관리한다.
- 색인마디블로크(inode block): 이 구역은 파일체계의 매 파일을 위한 표를 포함한다. 파일과 등록부의 모든 속성은 파일이나 등록부 그자체의 이름을 제외하고는 이 구역에 보존된다.
- 자료블로크(data block): 조작체계파일이나 사용자에 의하여 만들어 진 모든 자료 및 프로그람들이 이 구역에 존재한다.

파일체계배치를 그림 21-2에서 보여 준다. 다음절들에서 핵심부와 이 구성요소들이 파일들을 위한 공간할당을 조직하기 위해 순차적으로 작업하는 방식을 구체적으로 보게 된다.

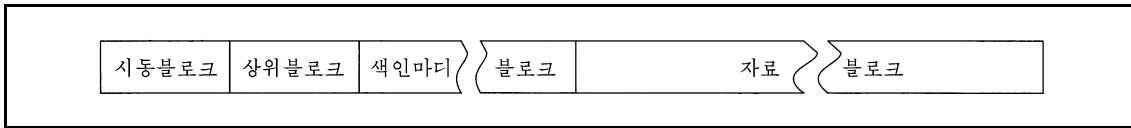



그림 21-2. 파일체계

그전에 먼저 이 파일체계의 장치이름체계(device naming scheme)에 대하여 설명한다.

주해

단어 《구획》과 《파일체계》는 자주 같은 뜻으로 리용되며 특히 구획이 하나의 파일체계를 포함할 때 더욱 그러하다. 그렇지만 이것은 항상 그러한것이 아니다. SCO UNIX와 같은 일부 체계들은 구획을 분할구역(division)들로 가르고 매 분할구역들이 하나의 파일체계를 가지게 한다. Linux는 확장구획(extended partition)을 만들수 있게(매 확장구획에 여러개의 론리구획이 존재) 한다. 이때 매 론리구획(logical partition)은 하나의 파일체계를 포함한다. 이와 같이 하나의 구획에 여러개의 파일체계를 가지는것이 가능하다.

21.4.3 파일체계용장치

SCSI디스크들은 보통 8개까지의 구획을 지원하며 IDE디스크들은 4개로 제한된다. 모든 파일체계는 각각의 장치이름을 가지며 이 이름들은 UNIX변종들에 따라 차이가 있다.

SVR4는 블로크장치와 문자장치를 각각 격납하는 등록부/dev/dsk와 /dev/rdsk를 리용한다. 또한 조종기번호,조종기상에서의 구동기번호와 구획번호를 포함하는 명명체계를 쓰기도 한다. 아래에 이 체계를 리용하는 Solaris체계상의 2개의 장치이름을 주었다.

- /dev/rdsk/c0t3d0s4-이것은 첫 조종기(c0)안에서의 첫 구동기(d0)를 위한 5번째 구획(s4)이다.
- /dev/rdsk/c0t3d0s2-이것은 디스크전체를 나타내는 3번째 구획(S2)이다.

구획들(또는 파일체계들)을 지정하는 장치들뿐아니라 UNIX체계는 디스크전체를 의미하는 어떤 특정한 장치도 가진다. Solaris에서 이 장치이름은 구획 2(3번째 구획)를 의미한다. 다른 체계들에서는 이름체계가 각이하다.

주해

어떤 구획이 여러 파일체계를 포함한다면 모든 파일체계와 구획은 그자체의 장치파일을 가지며 하드디스크전체를 위하여 어떤 별도의 장치도 있다.

Linux

파일체계를 위한 장치

Linux는 더 간단한 명명구조를 가진다. 기초파일이름으로서 SCSI디스크들은 접두사 sd를 가지며 IDE디스크들은 hd를 가진다. 그뒤에 구동기문자와 구획번호가 놓인다.

```
/dev/hda1-첫번째(a) IDE하드디스크의 구획1
/dev/sdb3-두번째(b) SCSI하드디스크의 구획3
/dev/hdb -두번째 IDE하드디스크전체
/dev/sda -첫번째 SCSI하드디스크전체
```

우에서 볼수 있는것처럼 Linux도 디스크전체를 의미하는 어떤 특정한 장치를 가진다. 이 장치이름은 파일이름체계에서 구획번호를 가지지 않는다.

/dev/hda1이 확장구획이고 2개의 론리구획을 포함한다면 어떻게 될것인가? 그 경우에 이 두개의 론리구획은 장치이름 /dev/hda5와 /dev/hda6을 가진다. 후에 설명하게 될 fdisk 출력을 보면 알수 있을것이다.

21.5 파일체계의 구성요소

파일체계의 4가지 구성요소들의 기능을 뒤에서부터 설명하기로 하자. 먼저 색인마디와 자료블로크를, 그다음 상위블로크와 기동블로크를 설명한다. 사실 순서가 뒤바뀐 학습방법이 파일체계의 작업을 더 잘 리해하도록 하여 준다.

21.5.1 색인마디블로크

모든 파일은 그 파일에 대한 거의 모든 정보(이름은 제외)를 포함하는 128byte의 표 즉 **색인마디**(inode)를 가지고 있다. 모든 색인마디들은 파일체계에서 사용자가 접근하기 어려운 구역안의 련속적인 **색인마디블로크**들속에 보관된다. 매 색인마디는 파일의 다음과 같은 속성들을 포함한다.

- 파일의 형(보통파일, 등록부, 장치 등)
- 련결의 수(파일이 가지고 있는 별명(alias)의 수)
- 소유자의 수값UID
- 소유자의 수값GUID
- 파일방식(3개의 허가권들의 쌍)
- 파일의 바이트수
- 파일자료의 최종변경날자와 시간
- 파일자료의 최종접근날자와 시간
- 색인마디의 최종변경날자와 시간
- 파일에 대한 15개의 지시자들의 배렬

색인마디는 **색인마디번호**(inode number)라고 불리우는 어떤 수값에 의하여 접근된다. 이 수값은 하나의 파일체계에서는 모든 파일들에 대하여 유일적이다. 그것은 실제상 이 색인마디블로크들속에서 그 색인마디의 위치이며 따라서 단순한 산수연산으로 그 위치가 정해 질수 있다. 앞에서 설명된바와 같이 색인마디번호는 ls의 -i선택항목으로 현시된다.

파일의 이름이나 색인마디번호는 그 색인마디안에 보관되지 않는다. 이 두 파라메터는 사실 그 파일을 수용하는 등록부안에 보관된다. 15개 지시자들의 배렬을 제외하고 이 파일속성들은 모두 ls의 여러가지 선택항목들에 의하여 표시될수 있다. ls지령이 파일이름을 현시하기 위해서는 그 등록부파일을 찾아보아야 한다.

어떤 파일이 열릴 때 그의 색인마디가 하드디스크로부터 주기억안에서 관리되는 체계자체의 색인마디표에로 복사된다. 핵심부는 항상 기억기사본(**색인마디완충기**: inode buffers)을 리용하여 작업하지만 정기적으로 기억기사본을 가지고 디스크사본을 갱신한다. 왜냐하면 두 사본이 동기화되지 않고(하나가 다른것보다 더 새것이므로) 기계의 전원이 비정상적으로 차단될 때 파일체계의 완전성이 손상되기때문이다. UNIX가 기동하려 하지 않을 때에는 파일체계검사를 진행해야 한다.

주해

색인마디는 파일이름을 제외한 파일의 모든 속성을 포함한다. 파일이름은 그 파일을 수용하는 등록부안에 보관된다. UNIX의 색인마디구조는 Windows FAT파일체계와 섬세한 대조를 이룬다(FAT체계는 변경시간, 바이트수, 디스크시작클라스터주소를 포함한 파일속성들을 체계구역자체의 등록부부분안에 보관한다).

21.5.2 자료블로크

거의 모든 체계들에서 디스크조종기에 의하여 읽기쓰기될수 있는 최소블로크는 512byte이다. 흔히 그것을 **물리블로크**(physical block)라고 부른다. 핵심부는 다른 블로크크기를 사용하여 자료를 읽거나 쓰는데 이 블로크를 흔히 **론리블로크**(logical block)라고 한다. 많은 UNIX도구들은 출력을 물리블로크단위로 통지하지만 체계의 능률은 파일체계를 만들 때 설정한 론리블로크의 크기에 의하여 결정된다.

한번에 한 문자씩 읽거나 쓰는 말단이나 인쇄기(문자장치)와 달리 하드디스크(블로크장치)들은 자료를 덩어리나 블로크단위로 처리한다. 표준(론리적)블로크크기는 1024로부터 8192(Solaris)까지의 범위에서 변화된다. 그러나 읽기쓰기조작을 위하여 블로크전체를 사용하는것은 1024byte의 디스크블로크에 3의 자료를 써야 할 때에 1021byte는 쓰기랑비로 될것이다.

자료블로크(data block)들은 색인마디블로크들이 끝나는 점에서부터 시작된다. 모든 블로크들은 자료블로크구역에서 블로크의 위치를 가리키는 번호 즉 주소에 의하여 식별된다. 자료를 포함하는 블로크들을 **직접블로크**(direct block)라고 한다.

블로크들이 련속적으로 번호가 매겨 져 있다고 해도 련속적인 블로크들안에 배렬된 자주 변경된 파일의 자료를 찾는다는것은 어려울것이다. 파일이 확장될 때 핵심부는 린접한 빈 블로크를 찾지 못할수도 있으며 디스크전반에 무질서하게 흩어 져 있는 블로크들로부터 빈것을 할당해야 한다. 이것은 읽기쓰기조작을 느리게 만들며 **디스크쪼각화**(disk fragmentation)를 초래한다. 그렇지만 이 쪼각화도 파일들이 커지거나 작아 지는것을 가능하게 한다.

이 모든것은 색인마디가 모든 직접블로크주소의 경로를 보관해야 한다는것을 의미한다. 그렇지만 색인마디는 그것들중 일부(12개)만을 보관할수 있다. 파일체계는 몇개의 **간접블로크**(indirect block)들도 포함한다. 이것들은 색인마디안에 수용할수 없는 직접블로크(13번째 블로크부터)들의 주소들만을 포함할뿐 자료는 포함하지 않는다. 그러나 색인마디는 이 간접블로크주소들의 목록을 관리하며 이 배렬이 파일에 리용되는 모든 블로크들의 경로를 어떻게 보관하게 하는가를 알수 있다.

주해

체계상의 론리블로크가 1024 혹은 8192byte의 크기를 가진다고 할지라도 많은 UNIX지령들은 출력통지에 물리블로크를 리용한다. 이 블로크는 512byte의 크기를 가진다. 거의 모든 체계들에서 ls -s, df, du, find지령들은 물리블로크만을 리해하며 물리블로크로만 통지한다.

21.5.3 블로크주소화체계

이 절에서는 Solaris에 의해 사용된 표준파일체계(ufs)를 참고로 하여 설명하지만 그 개념들은 일반성 있게 적용될수 있다. 색인마디안에 있는 15개의 디스크블로크주소들의 배렬이 어떻게 되여 파일의 자료블로크들의 경로를 보관하는데 충분한지는 알수 있을것이다.

첫 12개의 항목(entry)은 그 파일의 처음 12개 블로크들의 주소들을 포함한다. 그러나 어떤 파일이 3개 블로크의 크기라면 첫 3개의 항목들만 리용되며 나머지항목들은 0으로 채워 진다. 13번째이상으로부터 3개의 항목은 직접블로크들을 지정하지 않으며 오직 간접블로크만을 지정한다.

13번째항목은 보다 직접적인 많은 블로크주소들을 포함하는 **단일간접블로크**(single indirect block)의 주소를 가진다. 파일크기가 더 증가될 때 14번째 항목은 **2중간접블로크**(double indirect block)를 지

정하는데 그것은 단일간접블로크의 주소들을 포함한다. 15번째와 마지막항목은 **3중간접블로크**(triple indirect block)를 지정하는데 이것은 2중간접블로크를 지정한다. 최종적으로 이 모든 간접블로크들은 파일에 리용된 모든 블로크들을 보관하기 위한 **련결목록**(linked list)을 형성한다. 자료블로크의 구성을 그림 21-3에서 보여 준다.

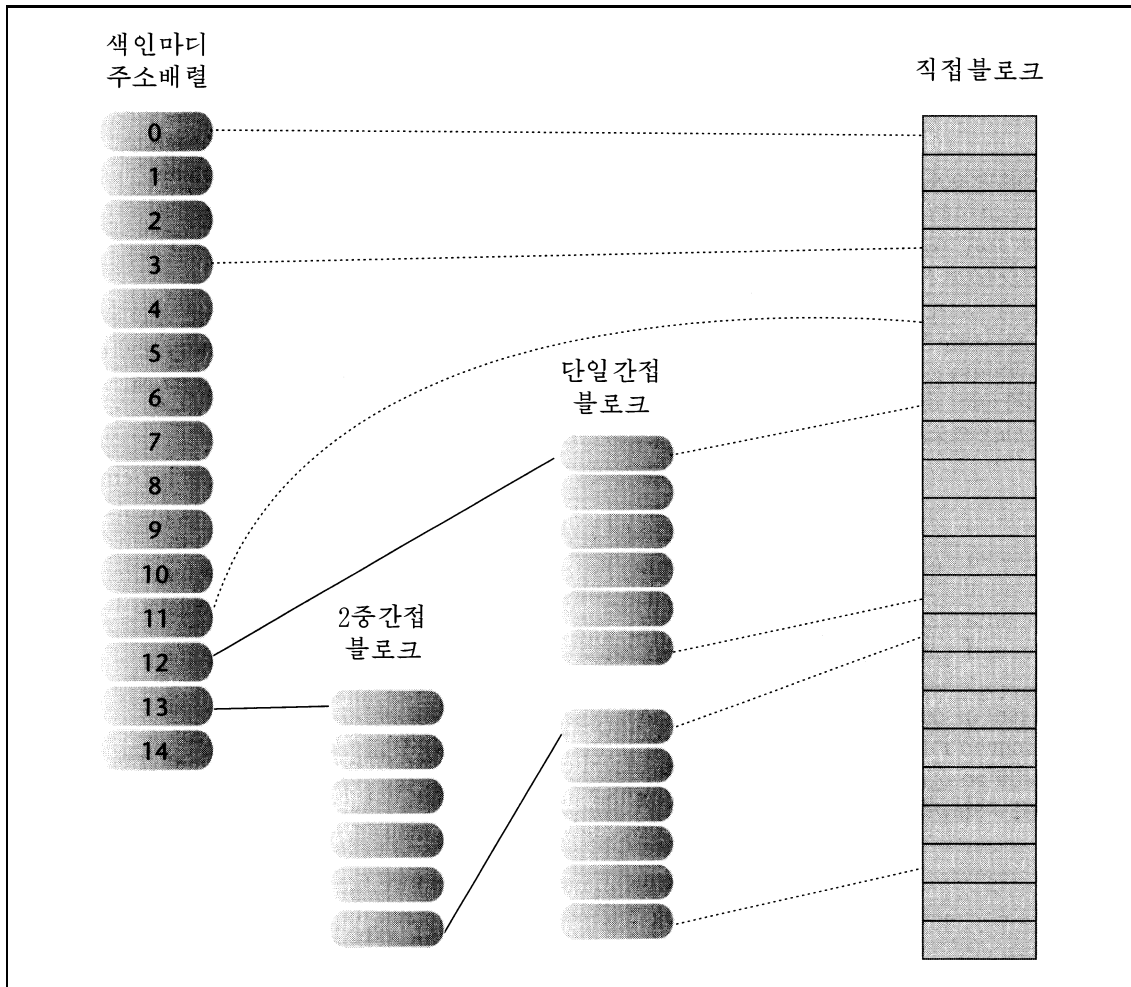



그림 21-3. 디스크블로크의 주소화도식(3중간접블로크는 제외)

주해 우에서 설명된 ufs파일체계에서의 한 파일의 최대크기는 32bit기계에서 2GB이다. 그후 UNIX는 64bit로 되였으며 수테라바이트(1TB = 1000GB)를 초과하는 파일크기를 가지는것도 가능하게 되였다.

21.5.4 상위블로크

색인마디블로크들의 앞에는 모든 UNIX파일체계의 《대차대조표(balanced sheet)》라고 하는 **상위블로크**(super block)가 놓인다. 그것은 자료블로크와 색인마디의 유용성과 디스크사용에 대한 전반적인 파일정보를 포함한다. 그러므로 체계의 안정한 조작을 위해서는 그 정보들이 정확해야 한다. 포함되는 정보들은 기본적으로 아래와 같다.

- 파일체계의 크기
- 파일체계의 론리블로크의 길이
- 갱신된 마지막시간

- 유효한 빈 자료블로크의 수와 즉시 할당할수 있는 빈 자료블로크들의 부분목록
- 유효한 빈 색인마디들의 수와 즉시 리용할수 있는 색인마디들의 부분목록
- 파일체계의 상태(《청결(clean)》 또는 《불결(dirty)》)

핵심부는 주기억안에서 색인마디와 함께 상위블로크의 사본도 관리한다. 핵심부는 색인마디와 자료블로크들의 할당을 조종할 때 이 사본을 읽거나 쓴다. 때때로 핵심부는 주기억사본으로 디스크사본을 갱신하기 위하여 sync조작을 사용한다. 이것은 디스크사본이 주기억사본보다 결코 더 새로울수 없으며 최신정보에 가깝게 될수만 있다는것을 의미한다. 이 처리는 기계를 끄기(shut down) 바로전에도 일어 난다.

주해 만일 상위블로크가 손상되면 UNIX는 기동하지 않는다. 이 문제를 해결하기 위하여 많은 체계들(Solaris나 Linux와 같은)은 디스크의 여러 구역에 상위블로크들을 써넣는다. 한 상위블로크가 파괴되면 체계가 또 다른 상위블로크를 사용하도록 지정할수 있다.

21.5.5 기동블로크

상위블로크보다 앞서 있는것이 **기동블로크**(boot block)이다. 이것은 Linux사용자들이 체계를 설치할 때 LILO선택항목으로서 보게 되는 **주기동레코드**(Master Boot Record:MBR)이다. 기동블로크는 구획표와 작은 초기적재프로그람(boot strapping)을 포함한다.

체계가 기동될 때 체계BIOS는 첫 하드디스크의 존재를 검사하며 기동블로크의 전체 토막을 주기억안으로 읽어 들인다. 그다음 초기적재프로그람에 조종권을 넘겨 준다. 다시 이것은 핵심부(파일 unix, genunix 혹은 vmlinuz)를 주기억으로 읽어 들인다. 여기서 초기적재프로그람은 뿌리(기본)파일체계의 기동블로크에서만 읽혀 지며 다른 파일체계에서는 이 블로크가 비여 있다.

21.6 등록부

등록부는 바로 2개의 파일속성 즉 파일이름과 색인마디번호를 포함한다. 색인마디 그자체는 이 번호에 의하여 접근될수 있기때문에 등록부와 색인마디는 2개의 표(색인마디번호를 통하여 서로가 관계되는)를 가진 어떤 관계형자료기지를 형성한다고 말할수 있다. RDBMS전문용어를 리용하면 이 두 표들의 각각의 행을 결합하여 파일의 속성들을 얻을수 있다.

어떤 파일에 대한 련결을 만들 때에는 그 색인마디안에서의 그의 련결계수기가 증가된다. 등록부항목(directory entry)도 원래의 파일항목으로부터 복사된 새로운 파일이름과 색인마디번호에 대하여 생성된다. rm을 리용하여 련결된 파일을 삭제할 때 색인마디안의 련결계수기가 감소되며 등록부항목도 그 련결을 삭제한다. 련결계수기가 0으로 떨어 지면 파일은 삭제된것으로 간주된다. 이때 해당 디스크블로크들은 해방되며 새로운 할당에 리용할수 있게 된다.

핵심부가 파일에 접근하는 방법

이제는 파일이나 등록부에 접근하기 위하여 핵심부가 사용하는 주소화기구(addressing mechanism)에 대하여 그리고 지령 cat foo를 줄 때 어떤 일이 일어 나는가를 알아 보자. 파일의 위치를 알아 내여 표준출력상에 현시할뿐아니라 핵심부는 파일끝에 도달했는가도 결정해야 한다. 작업의 많은 량이 여기에 관계된다.

1. 핵심부는 항상 현재등록부를 위한 색인마디번호를 주기억안에서 관리한다. 이 번호를 사용하여

색인마디블로크들을 탐색하며 이 등록부를 위한 색인마디의 위치를 알아 낸다.

2. 이 색인마디로부터 그 등록부파일을 포함하는 자료블로크의 주소를 얻어 낸다.
3. 핵심부는 등록부파일로부터 파일 foo와 그의 색인마디의 위치를 알아 낸다.
4. 그다음 색인마디블로크들로 돌아 가서 foo를 위한 색인마디의 위치를 알아 낸다.
5. 이제 핵심부는 파일크기와 디스크주소항목들을 읽고 간접블로크에로 가서 모든 관계되는 직접블로크들을 읽는다.
6. 마지막으로 디스크자두를 각각의 블로크들에로 움직이도록 디스크구동기에 지시하고 읽혀 진 바이트수를 계수하여 파일크기와 비교하며 두 수가 맞을 때까지 읽어 들인다. 파일끝에 도달하였는가를 알아 내는 다른 방법은 없다.

참고

하나이상의 블로크들(불련속일수도 있는)을 차지하는 큰 등록부들을 만들어 핵심부의 부담을 증가시키지는 말아야 한다. 그것은 파일접근을 느리게 할뿐이며 그대신 다른 등록부를 만드는것이 더 좋을것이다.

21.7 표준파일체계

앞에서 설명한것들은 단일파일체계에 대한것이다. 실생활에서 거의 모든 UNIX체계들은 디스크를 8개정도의 많은 구획으로 가른다. 더우기 여러개의 디스크를 가지고 있는 경우에는 그 모든 디스크들이 적어도 하나의 파일체계를 가져야 한다. 보통 대부분의 UNIX체계들이 항상 다음과 같은 2개의 파일체계를 가지게 된다.

- 뿌리파일체계(root file system): 이 파일체계는 모든 UNIX체계들에 반드시 존재해야 한다. 뿌리는 곧자 UNIX 즉 뿌리등록부, /bin, /usr/bin, /etc,/sbin, /usr/sbin, /dev, /lib등록부와 체계가동을 유지하는데 적절한 모든 편의프로그람들을 포함한다. 체계가 단일사용자방식으로 기동할 때 이것은 단지 체계관리자에게만 유효한 파일체계이다.
- 교체파일체계(swap file system): 모든 체계는 핵심부가 프로쎄스들의 이동을 조종하기 위해 사용하는 교체파일체계를 가진다. 체계기억기가 무겁게 읽혀 질 때 핵심부는 주기억안의 프로쎄스들을 이 파일체계에로 이동시켜야 한다. 이 교체된 프로쎄스들이 실행될 준비가 되면 주기억에 다시 읽혀 진다. 사용자들은 직접 이 파일체계안에 있는 자료에 접근할수 없다.

어떤 UNIX체계(특히 대규모의 체계인 경우)는 보다 많은 파일체계들을 포함한다. 체계파일들은 사용자가 만든 자료파일들과 분리되여 보관되여야 하며 따라서 그것들을 위한 분리된 파일체계가 만들어진다. Linux와 SVR4는 /home안에 사용자들의 홈등록부를 만들지만 이전의 체계들은 /usr를 리용한다. /home은 종종 분리된 파일체계로서 관리된다. 만일 /home상에 남은 공간이 없다면 관리자는 /usr2이나 /u와 같은 다른 파일체계를 리용한다.

림시파일이나 log파일들이 무제한 커질수 없도록 /tmp나 /var/tmp와 같은 추가적인 파일체계들을 가질수 있다. 만일 체계가 많은 량의 우편을 관리하고 있다면 /var/mail을 분리된 파일체계로서 가지도록 한다. 관리자가 자료기지쏘프트웨어를 설치하려고 하는 경우는 다른 파일체계들도 있을수 있다. 실례로 Oracle RDBMS를 위해서 /oracle가 있을수 있다.

21.8 파일체계의 형

파일체계는 처음 AT&T와 버클리가 만든 두가지 형태뿐이였다. 시간이 흐름에 따라 더 많은 파일체계형들이 UNIX체계안에 등록되였다. 자주 맞다들리는 파일체계들은 다음과 같다.

- s5
  SVR4가 나오기전에 System V에서 사용한 유일한 파일체계였으나 오늘날에는 SVR4가 역방향호환성을 위하여 이 이름을 제공할뿐이다. 이 파일체계는 512 혹은 1024byte의 론리블로크크기와 단일상위블로크를 사용한다. 14문자이상의 긴 파일이름은 관리할수 없다.
- ufs
  거의 모든 UNIX체계들에 적용된 파일체계이다. 블로크크기가 64KB로 늘어 났기때문에 이 파일체계의 능률은 s5보다 더 좋다. 상위블로크를 보관하는 원통묶음을 가진 다중상위블로크를 사용한다. s5와 달리 ufs는 255문자의 파일이름, 기호련결(symbolic link)와 디스크할당(disk quota)을 지원한다.
- ext2
  Linux의 표준파일체계이다. 1024byte의 블로크크기를 사용하며 ufs와 같이 다중상위블로크와 기호련결을 사용한다.
- iso9660 혹은 hsfs
  CD-ROM에 사용된 표준파일체계이며 DOS형의 8+3파일이름을 리용한다. UNIX가 더 긴 파일이름을 사용하면서부터 hsfs도 그것들을 수용하기 위한 Rock Ridge확장자를 제공한다.
- msdos 혹은 pcfs
  대부분의 UNIX체계들도 DOS파일체계를 지원한다. Windows체계상에서 사용하기 위해 플로피디스크상에 이 파일체계를 만들고 파일들을 전송할수 있다. Linux와 Solaris도 하드디스크안의 DOS파일체계를 직접 접근할수 있다.
- swap
  이 파일체계는 이미 해설되였다.
- bfs
  기동파일체계(boot file system) SVR4가 초기적재프로그람과 UNIX핵심부를 주관하기 위해 사용한다. 사용자들은 이 파일체계를 사용할 필요가 없다.
- proc 혹은 procfs
  주기억안에서 관리되는 모조파일체계(pseudo-file system)로 간주할수 있다. 매 실행프로쎄스의 자료를 보관하며 파일들을 포함하는것처럼 보이지만 실제적으로는 아무것도 포함하지 않는다. 사용자들은 PID를 포함한 거의 모든 프로쎄스정보를 여기에서 직접 얻을수 있다.

이 파일체계들의 일부는 어떤 독특한 특징을 가지지만 그것을 무시하도록 하자. UNIX제공자들도 자체의 파일체계를 가지지만 그것들도 우에서 설명한 대부분의 형태를 지원한다. 파일체계를 조작하는 지령들(mkfs와 mount와 같은)은 파일체계를 서술하는 선택항목을 사용하며 따라서 사용하고 있는 파일체계를 알아야 한다.

21.9 구획과 파일체계의 만들기

구획을 만들게 되면 UNIX쪼각화도 아주 명백해 진다. UNIX의 여러가지 판본들은 디스크를 가르는데 여러가지 도구를 리용한다. 그 원리는 대체로 류사하지만 때때로 구체적인 차이점들이 그 류사성을 보잘것 없는것으로 만든다. 체계관리자들은 대체로 자기의 체계상에서 구획을 만들거나 삭제하는것을 허용하지 않으므로 Linux체계상에서 그것들을 만들것이다.

21.9.1 fdisk를 리용한 구획의 만들기

Linux나 SCO UNIX는 다같이 사용자가 인텔콤퓨터상에서 다중조작체계를 소유할수 있도록 허락한다. 두 체계가 구획의 생성, 삭제, 활성화에 Windows형의 fdisk지령을 제공한다는것은 놀라운 일이 아니다.

이제 Oracle RDBMS쏘프트웨어를 채용하기 위한 구획을 첫 IDE하드디스크상에 만들어 보자. fdisk를 실행시키고 그다음 p지령을 사용하여 구획표를 보자.

```
Disk /dev/hda: 255 heads, 63 sectors, 784 cylinders
Units = cylinders of 16065 * 512 bytes

Device    Boot  Start      End   Blocks    Id  System
/dev/hda1    *      1      217   1743021    5  Extended
/dev/hda2         218      478   2096482+   6  DOS 16-bit >=32M
/dev/hda4         620      784   1325362+  83  Linux native
/dev/hda5           1        5     40099+  82  Linux swap
/dev/hda6           6      217   1702858   83  Linux native
```

이 디스크는 16065×512byte크기의 원통 784개를 포함한다. 이것은 몇개의 구획을 포함하는 6.5GB 디스크(784×16065×512)이다. Linux의 구획들은 DOS의 경우와 더 류사하게 동작한다. 이전 상태처럼 하나의 파일체계를 가진 기본구획(primary partition)을 만들거나 여러개의 론리구획을 포함하는 확장구획을 만들수 있다.

구획1(hda1)이 구획5(hda5) 및 6(hda6)과 꼭 같은 원통들을 사용하고 있다. 이 구획(hda1)은 실지로 이 두개의 론리구획을 포함하는 확장구획이다. 그것들중 하나는 체계의 **교체구획**(swap partition) hda5을 포함하며 다른 하나는 Linux체계(hda6)를 포함한다. 매 구획의 크기는 Blocks밑에 현시되는데 한 블로크는1024byte이다. 능동구획(active partition)은 2번째 렬에 별표(*)로 표시한다.

이 디스크상에 다른 조작체계들이 존재하는것을 볼수 있다. 구획4(hda4)우에 또 다른 Linux체계가 있으며 구획2(hda2)상에는 DOS파일체계(Windows를 실행시키는)가 있다. hda5와 hda6은 hda1의 부분으로 간주될수 있으므로 여기에는 3개의 웃준위(top-level)구획이 있다. 또한 479와 619사이의 원통들이 사용되지 않고 있으며 그 공간을 어떤 구획을 만드는데 쓸수 있다. 이것은 구획3이 될것이다.

fdisk m지령은 자기의 모든 내부지령들을 보여 주는데 그중 다음의 지령들이 우리의 목적을 실현하는데 리용된다.

```
Command (m for help): m
Command action
```

```
a   toggle a bootable flag
d   delete a partition
l   list known partition types
m   print this menu
n   add a new partition
p   print the partition table
q   quit without saving changes
w   write table to disk and exit
```

구획을 만들기 위해서는 n지령을 사용하며 구획번호와 시작원통번호, 끌원통번호를 주어야 한다.

```
Commamd(m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
p
Partition number (1-4): 3
First cylinder (479-784, default 479): 479
Last cylinder or +size or +sizeM or +sizeK (479-619, default 619): 619
```

Linux는 초기원통값들을 재치 있게 보여 주지만 값들을 명백히 입력하는것이 좋다. 이제는 p를 사용하여 새로운 구획을 보도록 하자.

```
   Device   Boot   Start    End   Blocks    Id  System
/dev/hda1     *        1    217   1743021    5  Extended
/dev/hda2            218    478   2096482+   6  DOS 16-bit >=32M
/dev/hda3            479    619   1132582+  83  Linux native              새로운 구획
/dev/hda4            620    784   1325362+  83  Linux native
/dev/hda5              1      5     40099+  82  Linux swap
/dev/hda6              6    217   1702858   83  Linux native
```

이제는 1GB이상의 공간을 차지하는 새로운 구획 hda3이 생겨 났다. 이제 w지령을 써서 수정된 구획표를 기동블로크에 써넣어야 한다. 체계를 재기동하겠는가 하는 질문을 받게 되며 체계는 새로운 파일체계를 유효하게 만들것이다.

주해

하나의 Linux구획안에 여러개의 파일체계를 가지려면 먼저 확장구획을 만들고 그다음 이 확장구획안에 여러개의 론리구획을 생성해야 한다. 만일 fdisk를 가지고 작업하는것이 어렵다고 생각되면 차림표기반프로그람 cfdisk를 리용할수도 있다.

21.9.2 mkfs를 리용한 파일체계의 만들기

구획을 만든 다음 그것을 사용할수 있게 하려면 이 구획상에 파일체계를 만들어야 한다. 앞단(front-end)도구들을 쓸수 있는데 그 도구들은 보통 보편적인 파일체계생성도구인 mkfs를 리용하여 조작을 진행한다. Linux에서 방금 만든 구획에 대하여 mkfs지령을 -t선택항목과 함께 써서 파일체계의 형을 지정하여 보자.

```
# mkfs -t ext2 /dev/hda3
mke2fs 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label =
283560 inodes, 1132582 blocks
56629 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
139 block groups
8192 blocks per group, 8192 fragments per group
2040 inodes per group
Superblock backups stored on blocks:
     8193, 16385, 24577, 32769, 40961, 49153, 57345, 65537, 73729, 81921,
     90113, 98305, 106497, 114689, 122881, 131073, 139265, 147457, 155649,
   .......
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

이것은 1024byte의 블로크크기를 사용하는 표준Linux파일체계인 ext2파일체계를 만든다. 균일(8192개의 블로크)하게 흩어 져 있는 여러개의 상위블로크들을 주의해야 한다. ufs와 ext2파일체계들도 장비를 최소화하기 위해 토막들을 사용한다.

Linux에서 mkfs는 실제적으로 파일체계를 생성하는 mke2fs의 앞단으로서 동작한다. 블로크의 번호을 지정하지 않았으므로 mkfs는 색인마디와 자료블로크의 번호에 초기값을 사용하였다. 다량의 매우 작은 파일들을 다루지 않는한 보통 초기값을 바꿀 필요는 없다.

참고

mkfs지령의 출력으로부터 상위블로크의 복사본을 포함하는 블로크주소들의 레코드를 보관하여야 한다. 기본상위블로크가 손상되게 되면 그때 fsck를 사용하여 상위블로크의 번호를 바꾸어야 한다.

주해

Solaris는 구획을 만들기 위해 차림표형식의 format지령을 사용한다. 구획차림표를 꺼내기 위하여 format>프롬프트에서 구획을 불러 내야 한다. 차림표는 8개까지의 구획들을 변화시킬수 있게하며 거기서 1과 그리고 번호가 붙은 구획(2번째와 3번째)들이 교체구획과 전체 구획으로 각각 예약된다. 디스크에 구획표를 써넣으려면 마지막으로 label지령을 사용해야 한다. Solaris(또는 HP-UX)도 역시 mkfs의 앞단으로서 newfs지령을 사용한다.

```
newfs /dev/rdsk/c0t3d0s0
```

21.10 파일체계의 태우기와 내리우기

이제는 파일체계가 빈 뿌리등록부와 색인마디구역을 가진 4개의 구성요소들을 가지게 되였다. 그렇지만 뿌리(기본)파일체계(설치시에 만들어 진)는 그의 존재를 알지 못한다. 이 새로운 파일체계는 기본

파일체계(hda4나 hda6일수 있다.)에 부속된후에만 자료를 보존할수 있다.

이러한 부속은 파일체계가 여러 점에서 기본파일체계에 자신을 부속시키는 프로세스 즉 **태우기**(mounting)에 의해 일어 난다. 이 련결이 놓이는 점을 **태우기점**(mount point)이라고 부른다. 태우기후에 뿌리파일체계는 《주》파일체계로 되며 그 뿌리등록부도 역시 통합된 파일체계의 뿌리등록부로 된다.

21.10.1 파일체계의 태우기(mount)

파일체계를 태우는데 mount지령이 쓰인다. 새로운 파일체계를 태울 때 그 지령은 2개의 인수 즉 그 파일체계의 장치이름과 태워 져야 할 등록부를 가진다. 파일체계를 태우기전에 먼저 기본파일체계안에 빈 등록부(례:/oracle)를 만들어야 한다. 새로운 파일체계의 뿌리등록부는 이 등록부상에 태워 지게 된다.

mount는 파일체계의 형을 지정하기 위한 선택항목을 사용한다. 이 선택항목은 UNIX변종에 따라 다르다. 아래에 Solaris와 Linux체계상에 있는 /oracle등록부에 파일체계를 태우기 위하여 mount지령을 사용하는 방법을 주었다.

```
mount -F ufs /dev/dsk/c0t3d0s5 /oracle          Solaris
mount -t ext2 /dev/hda3 /oracle                 Linux
```

장치를 태운후에 mkfs로 만든 파일체계의 뿌리등록부는 구분된 독자성을 잃어 버린다. 이제는 등록부 /oracle로 되며 기본파일체계의 부분인것처럼 보이게 된다. 태우기의 마지막결과는 사용자가 단일한 파일체계를 보게 된다는것이며 /oracle로 부터 /home으로 이동하는 파일이 실제적으로는 두 하드디스크사이에서 움직일수도 있다는것을 생각하지 않아도 된다는것이다.

앞에서 태워 진 파일체계는 초기의것이였으므로 -F(혹은 -t)선택항목이 중복되였다. 그렇지만 다른 파일체계를 태운다면 그 선택항목이 사용되여야 한다.

```
mount -F hsfs -r /dev/dsk/c0t6d0s0 /cdrom       CDROM-Solaris
mount -F pcfs /dev/diskette /floppy             DOS diskette-Solaris
mount -t iso9660 /dev/cdrom /mnt/cdrom          CDROM-Linux
mount -t vfat /dev/hda1 /msdos                  Windows hard disk-Linux
mount -t msdos /dev/fd0 /floppy                 DOS diskette-Linux
```

mount -a는 구성파일(configuration file) mount에 렬거된 모든 파일체계들을 태우며 mount만 사용되였을 때는 태워 진 모든 파일체계들을 렬거하여 보여 준다.

```
# mount
/ on /dev/dsk/c0t0d0s0 read/write/setuid/largefiles on Thu Apr 20 10:00:10 2000
/proc on /proc read/write/setuid on Thu Apr 20 10:00:10 2000
/dev/fd on fd read/write/setuid on Thu Apr 20 10:00:10 2000
/oracle on /dev/dsk/c0t0d0s3 setuid/read/write/largefiles on Thu Apr 20 10:00:15 2000
/u01 on /dev/dsk/c1t3d0s1 setuid/read/write/largefiles on Thu Apr 20 10:00:15 2000
/u02 on /dev/dsk/c1t3d0s3 setuid/read/write/largefiles on Thu Apr 20 10:00:15 2000
/u03 on /dev/dsk/c1t3d0s4 setuid/read/write/largefiles on Thu Apr 20 10:00:15 2000
```

mount는 이 정보를 /etc/mnttab(Solaris)나 /etc/mtab(Linux)로부터 가져 온다. mount의 구성파일의 내용은 간단히 설명하도록 한다.

Linux

Linux는 8×3형식의 파일이름을 가지는 단순한 DOS파일체계(msdos)와 긴 파일이름을 수용하는 Windows95/98/2000형의 파일체계(vfat)를 구별한다. 그러므로 태워진 DOS디스크안의 어떤 파일이름에서 ~(물결표)를 발견한다면 그때는 틀림없이 vfat를 msdos로 틀리게 태운것이다.

주해

태우기점은 보통 빈 등록부이지만 만일 일부 파일들을 포함한다면 이 파일들은 태운후에 보이지 않게 될것이다. 그 파일체계가 내리워 질 때 파일들이 다시 나타난다.

21.10.2 파일체계의 내리우기(umount)

내리우기(unmounting)는 umount(글자에 주의할것)지령으로 수행하며 인수로서 파일체계이름이나 태우기점을 요구한다. 앞에서 만들어 태운 파일체계에 대하여서는 다음의 첫 두 지령들중 어느것을 사용하든지 그 파일체계를 내리울수 있다.

```
umount /oracle
umount /dev/hda3
umount /dev/dsk/c0t3d0s5
```

파일체계안의 어떤 파일이 열려 있는 경우는 그 파일체계를 내리우는것이 불가능하다. 어떤 등록부를 삭제하려면 사용자가 그우의 등록부에 있어야 하는것처럼 어떤 파일체계를 내리우려면 그우에 위치해야 한다. 만일 그렇지 않으면 다음과 같은 오유통보가 나타난다.

```
# umount /dev/hda3
umount: /oracle: device is busy
```

또한 모든 파일체계 혹은 형에 따르는 일부 파일체계들을 내리울수도 있다.

```
umount -t ext2
umount -a
```

-a선택항목은 체계가동에 요구되는것들을 제외한 현재 태워 진 모든 파일체계들을 내리운다. 태워진 파일체계의 목록은 /etc/mtab(Linux) 혹은 /etc/mnttab(Solaris)에서 리용될수 있다.

21.10.3 mount선택항목들(-o)과 /etc/fstab

mount는 여러개의 특별한 선택항목들과 함께 실행될수 있으며 그 선택항목들앞에 -o선택항목이 먼저 놓인다. 이 선택항목들중 대부분은 파일체계전용이다. 이 선택항목들은 반점으로 구획 지어 진다. 비록 그것들이 Linux장치이름과 함께 쓰일지라도 거의 모든 선택항목들은 보편적으로 적용될수 있다.

```
mount -o ro /dev/sdb3 /usr/local
mount -o exec /dev/cdrom /mnt/cdrom
mount -o rw,remount /dev/hda3 /home
```

처음것은 파일체계를 읽기전용방식으로 태운다. mount는 그것을 위한 동의어 즉 -r선택항목을 가진다. 둘째것은 실행가능파일들을 CD-ROM으로부터 직접 실행시킬수 있게 한다. 어떤 파일체계를 읽기전용으로 태웠다면 그것을 읽기/쓰기방식으로 다시 태우기 위해 먼저 그것을 내리울 필요는 없다. rw와 remount선택항목을 함께 사용하면 된다.

mount는 구성파일을 사용하며 mount -a를 사용하면 그 파일안에 렬거된 모든 파일체계들이 순서대로 태워 진다. 체계가 시작될 때 같은 지령이 실행되며 그러므로 기계상에 태워 진 파일체계들을 항상 볼수 있다. 이 파일은 일반적으로 /etc/fstab이지만 Solaris는 다른 형식을 가진 SVR4형의 /etc/vfstab를 사용한다. 선택항목에도 일부 약간의 차이가 있다. Linux체계상의 /etc/fstab의 몇개 행을 보자.

```
# mount device  Mount point   File System Type  mount Options
/dev/hda5       swap          swap              defaults        0     0
/dev/hda6       /             ext2              defaults        1     1
/dev/hda2       /dosc         vfat              defaults        0     0
/dev/hda3       /oracle       ext2              defaults        1     2
/dev/hdc        /mnt/cdrom    iso9660           ro,noauto,user  0     0
/dev/fd0        /floppy       auto              noauto,user     0     0
none            /proc         proc              defaults        0     0
```

매행은 단일파일체계의 태우기특성을 제공한다. 목록은 플로피구동기와 CD-ROM도 포함한다. 첫 4개 렬은 읽기가 간단하다. noauto선택항목은 mount -a가 사용되였을 때에도 그 파일체계가 태워 지지 않도록 한다. 이 파일체계는 수동으로 태워야 한다.

/etc/fstab안에 CD-ROM(혹은 기타 장치)을 위한 태우기특성을 가지고 있다는것은 장치이름이나 태우기점을 인수로 하여 mount를 사용할수 있다는것을 의미한다.

```
mount /dev/hdc
mount /mnt/cdrom
```

user선택항목은 일반사용자들이 그 지령을 줄수 있도록 한다는것은 의미한다. 플로피장치를 위한 auto파일체계형은 mount가 /proc/filesystems를 보고 거기에 렬거된 모든 형들을 해보도록 한다는것을 의미한다.

5번째 마당의 1은 그 파일체계가 dump지령으로 여벌복사(backup)되여야 한다는것을 가리킨다. 6번째 마당은 체계기동시에 그 파일체계들이 fsck지령에 의하여 검사되기 위한 순서를 보여 준다.

주해

mount -a는 /etc/fstab 또는 /etc/vfstab(Solaris)안의 모든 파일체계들을 태우며 그것들은 기동시에 태워 질수도 있다. 마찬가지로 umount -a는 /etc/mnttab(Solaris) 혹은 /etc/mtab(Linux)안에 들어 있는 정보를 리용하여 이 모든 파일체계들을 내리운다. Solaris와 같은 일부 체계들은 모든 파일체계들을 태우거나 내리우기 위하여 mountall과 umountall지령(사실은 쉘스크립트)을 사용하기도 한다.

21.11 파일체계검사(fsck)

기억기복사본에 의한 디스크상위블로크와 색인마디블로크의 갱신을 지연시키는 UNIX의 내부특징은 많은 파일체계령역에서 모순을 만든다. 상위블로크가 디스크에 씌여 지기전에 전원이 차단되면 파일체계는 일관성을 잃는다. 파일체계를 손상시키는 모순점들이 많으며 가장 일반적인것들을 아래에서 보여 준다.

- 같은 디스크블로크를 요구하는 2개이상의 색인마디.
- 빈것으로 표시되여 있으나 상위블로크에 렬거되지 않은 블로크.
- 빈것으로 표시된 사용중의 블로크.

- 빈것으로 표시된것도 아니고 사용중인것도 아닌 색인마디 또는 범위를 벗어 나는 불량블로크번호를 가진 색인마디.
- 색인마디안에 지정된 파일크기와 주소배렬속에 지정된 자료블로크번호의 부적당한 짝.
- 잘못된 일람자료를 포함한 손상된 상위블로크.
- 등록부항목을 하나도 가지지 않거나 색인마디에 지정된 무효한 파일형을 가지는 파일.

fsck(파일체계일관성검사: file system conisistency check)지령은 손상된 파일체계를 검사하고 보관하는데 쓰인다. 그 지령은 실제적으로 작업을 하는 파일체계전용프로그람(fsck_ufs나 fsck.ext2와 같은)의 앞단으로 동작한다. 보통 파일체계의 설치가 실패하면 이것은 실행된다. Solaris를 포함하는 많은 체계들에서 파일체계들은《불결(dirty)》 혹은《청결(clean)》로 표식된다. fsck는 다음번 기동시에 불결한 파일체계만을 검사한다. /ect/fstab(Solaris에서는 /ect/vfstab)안에 렬거된 모든 파일체계들을 뿌리파일체계로부터 시작하여 6번째 마당에 지정된 순서로 검사하는것도 fsck에 지시한다. 이 지령은 파일체계의 이름을 인수로 하여 사용될수도 있다.

```
# fsck /dev/rdsk/c0t3d0s5
** /dev/rdsk/c0t3d0s5
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
```

fsck는 5개 단계로 검사를 수행하며 우의 출력은 파일체계가 모순이 없게 되였을 때 얻어 지는것이다. 그렇지만 파일체계가 손상되면 체계화면상에 통보문과 질문들이 나타나며 거기에 정확히 대답해야 한다. 아래에 fsck가 진행하는 매 단계를 설명한다.

- 1단계: 불량블로크와 중복블로크들의 형식과 블로크번호를 정정하기 위하여 그 색인마디들을 유효하게 한다. 블로크번호가 범위를 벗어 나면 그 블로크를 BAD로, 또 다른 색인마디에서 요구한다면 DUP로 선언한다.
- 2단계: 1단계에서 검출한 OUT OF RANGE 색인마디번호들을 위하여 뿌리로부터 시작하여 모든 등록부항목들을 검사한다. fsck는 그 파일이든가 등록부전체를 삭제하는것으로 그 오유를 정정한다.
- 3단계: 참고되지 않는 등록부들을 찾아 보고 다음실험을 위해 /lost+found안에 그 파일들을 보관한다. 여기서 파일들은 자기의 색인마디번호다음에 이름이 붙는다. 이 등록부가 모든 파일체계에 대하여 유용한가를 항상 확인해야 한다. 왜냐하면 fsck가 그것을 생성하지 않기때문이다.
- 4단계: 등록부항목을 가진 색인마디안에 축적된 련결계수(link count)를 검사하고 일어 난 손상의 정도에 의존하여 파일의 삭제 혹은 재련결(/lost+found등록부에로)을 문의한다. 그다음 fsck는 자기가 계산한 빈 색인마디계수값과 상위블로크안에 저축된 값을 비교한다.
- 5단계: 최종적으로 fsck의 빈 블로크계수값을 상위블로크안에서 관리되는 값과 비교한다.《구조(salvage)》조작은 사용자의 동의와 함께 수행될수도 있으며 오유가 있는 빈 블로크목록을 새로 계산된것으로 교체할것이다.

fsck가 선택항목이 없이 사용되였을 때에는 검출된 손상을 회복하기전에 먼저 문의한다. 보통 모든 질문에 y로 대답하는것이 안전하다. fsck -y(Linux에서 e2fsck는 -p를 사용한다.)는 모든 대답이 긍정적이라고 가정하고 어떤 응답도 기다리지 않고 처리한다. Solaris체계상에서 fsck -m는 교정을 진행하지 않고 단지 결함을 검출하기만 한다.

fsck는 블로크장치와 문자형장치들에서 작업하지만 문자장치상에서의 검사가 더 빠르다. 관리자는 항상 파일체계들을 내리운 조건에서 이 검사를 수행해야 한다. 뿌리파일체계는 내리워 질수 없으므로 이 파일체계상에서의 검사는 단일사용자방식에서 수행해야 한다.

만일 상위블로크가 손상되였다면 그때는 교체될 상위블로크번호를 지정하기 위하여 fsck를 -b선택항목(Solaris에서는 -o b=n,여기서 n은 블로크번호)과 함께 사용해야 한다. 이 목록은 그 파일체계가 초기화될 때 mkfs에 의하여 현시된다(Solaris에서는 newfs -Nv가 그것들을 현시한다). 파일체계는 흔히 교정할수 없게 손상되며 이때에는 체계의 재설치가 유일한 방도로 된다.

참고

fsck는 자주 일관성에서 이상하다고 간주되는 파일들의 색인마디번호들을 출력한다. 앞으로의 조사를 위하여 이 번호들에 주의를 돌려야 한다. 체계폭주후에 없어 진 파일들은 /lost+found 등록부에서 찾아 볼수 있다.

sync의 문제

update데몬은 주기억에 보관된 상위블로크, 색인마디, 자료블로크들을 꺼내기 위해 30초마다 sync를 호출한다. 그 지령을 손으로 입력하면 쓰기가 예정되였다는것을 알리여 프롬프트가 귀환되며 두번째 sync는 첫번째것이 끝나기전에는 시작되지 않는다.

fsck가 파일체계를 재구성할 때 상위블로크와 다른 표들의 기억기사본이 때때로 낡고 부정확한 정보를 포함할수도 있다. 이러한 비정상적인 경우에는 디스크상의 정보가 기억기사본보다 더 최근의것으로 된다. 그때 fsck는 다음의 통보를 내보낼수도 있다.

```
***** BOOT UNIX (NO SYNC!) *****
```

이것은 뿌리파일체계가 일련의 문제를 포함하고 있을 때 일어난다. 그 통보문은 만일 디스크에 부정확한 상위블로크를 써넣으려고 sync나 /sbin/shwtdown을 사용하면 fsck로서 수행한 모든 정상작업들이 류실되게 된다는것을 알려 준다. 그때에는 sync를 사용하지 말고 재설정단추를 즉시 눌러 체계를 재기동해야 한다. 이것은 다음의 sync조작이 디스크에 부정확한 정보를 자동적으로 쓰기전에 수행되여야 한다.

요 약

모든 장치파일들은 /dev안에 보관된다. 장치파일의 색인마디는 한조의 수(기본수와 보조수)를 포함한다. 기본수(major number)는 장치구동기 즉 장치의 형을 표현한다. 보조수는 핵심부가 장치구동기(device driver)에 보내는 파라메터를 표현하는 계렬번호이다.

장치들은 자료를 읽거나 쓰는 방법에 따라 블로크형(block-special) 혹은 문자형(character-special) 장치일수 있다.

SVR4는 블로크장치를 위해서는 /dev/dsk등록부를, 문자장치를 위해서는 /dev/rdsk등록부를 사용한다. 같은 장치가 역방향호환성이나 특수한 기능을 위해 다른 파일이름에 의하여 접근될수 있다.

하드디스크는 동심원모양의 경로들의 원통으로 표현되는 여러개의 표면으로 구성된다. 자료들은 이

경로안에 있는 기록분구에 씌여 진다.

UNIX파일체계는 보통 여러개의 파일체계들로 구성되며 매 파일체계는 자체의 뿌리등록부와 장치파일을 가진다. 하나의 파일체계는 하나의 분리된 구획에 수용된다. 분할은 자료의 쪼각화를 제한하며 한 구획의 자료가 다른 구획에 영향을 주지 않는다는것을 담보한다. 하드디스크전반을 표현하는 어떤 장치파일이 있다.

모든 파일체계들은 기동블로크, 상위블로크, 색인마디와 자료블로크들로 이루어 진다. 기동블로크는 기동정보와 구획표를 포함한다. 상위블로크는 빈 색인마디들과 빈 자료블로크들에 대한 상세한 정보를 포함하여 파일체계에 대한 전반적인 정보를 포함한다. 주기억안의 상위블로크는 sync에 의하여 주기적으로 디스크에 씌여 진다. 많은 체계들이 디스크의 여러 구역에 여러개의 상위블로크들을 관리한다.

색인마디는 파일이름을 제외한 모든 파일속성을 포함한다. 그것은 15개의 주소로 된 배렬을 포함하며 처음 12개는 첫 12개의 자료블로크의 주소를 보관한다. 다음의 3개는 파일에 사용된 나머지 디스크블로크번호들을 수용하는 련결목록을 형성한다. 상위블로크와 같이 주기억안의 색인마디도 역시 동기화되여 디스크에 씌여 진다.

등록부는 색인마디번호와 파일이름만을 포함하며 색인마디번호로서 파일의 색인마디와 관계된다. 한 파일이 련결될 때 그의 색인마디계수값은 1만큼 증가된다. 계수기가 0으로 떨어 지면 파일이 삭제된것으로 간주된다.

모든 UNIX체계는 교체파일체계와 뿌리파일체계를 가지게 된다. 대부분의 체계들은 오늘날 다중상위블로크, 기호련결, 디스크할당을 허용하는 ufs파일체계를 사용한다. Linux는 ext2파일체계를 사용한다. CD-ROM(hsfs 또는 iso 9660)과 DOS디스크(pcfs, vfat, msdos)를 위한 파일체계형들이 있으며 처리를 위한 모조파일체계(proc혹은 procfs)가 있다.

일부 체계들에서는 구획을 만들고 삭제하고 활성화하기 위해 fdisk지령을 사용한다. Linux체계들은 하나의 확장구획이 여러개의 론리구획을 수용하도록 하는것을 가능하게 한다. 파일체계를 만들기 위하여 mkfs가 사용되며 자주 앞단도구로서 쓰인다.

매 파일체계는 태우기(mount)되여야만 뿌리파일체계가 그것을 인식할수 있다. mount는 /etc/fstab(Solaris에서는 /etc/vfstab)로부터 태우기지시를 받는다. 사용자가 태우기점의 웃위치에 있지 않는 한 그리고 그안의 파일이 누군가에 의해 사용되고 있는한 그 파일체계는 내리워 질수 없다. 파일체계들은 내리워 질수도 있다(umount).

상위블로크와 색인마디들의 기억기사본이 체계를 끄기전에 디스크에 씌여 지지 못하면 파일체계파괴가 일어 난다. fsck는 파일체계의 일관성을 보통 내리워 진 조건에서 검사한다. 그것은 파일을 삭제하든지 혹은 련결되지 않은 파일들을 /lost+found에로 옮기든지 하여 불일치를 정정한다. 뿌리파일체계는 단일사용자방식에서만 검사될수 있다.

shutdown지령은 sync를 사용하여 상위블로크의 기억기내용을 파일체계에 써넣는다. 때때로 디스크상태가 더 새로울 때에는 동기화가 수행되지 말아야 한다.

시험문제

1. System V와 Linux에 의해 사용되는 플로피의 일반적인 장치이름은 무엇인가?
2. 기동가능한(bootable) 3.5인치 1.44MB플로피구동기의 장치이름은 무엇인가?
3. Linux체계에서 확장구획의 개념을 설명하시오.

4. 색인마디안에 보관되지 않는 유용한 파일속성은 어느것인가? 그것은 어디에 보관되는가?
5. 파일이 삭제될 때 색인마디안에서 어떤 변화가 일어 나는가?
6. 빈 색인마디와 자료블로크들의 수는 어디에 보관되는가?
7. SVR4와 Linux체계상에서 핵심부는 어느 등록부에 보관되는가?
8. 파일의 색인마디번호는 어떻게 찾아 내는가?
9. IDE와 SCSI하드디스크에 몇개의 구획을 가질수 있는가?
10. 파일체계를 만들기 위하여 앞단쏘프트웨어프로그람들이 어느 지령을 불러 내는가?
11. 파일체계를 내리우는것이 언제 불가능한가?
12. fsck는 접속되지 않은 파일에 대하여 보통 무엇을 하는가?
13. 뿌리파일체계에서 fsck를 보통 어떻게 실행시키는가?

련습문제

1. 장치의 기본번호와 보조번호란 무엇인가?
2. 어떤 등록부가 디스크공간을 전부 채우고 사용할 경향이 있다면 파일체계구성을 어떻게 수정해야 하는가?
3. Solaris에서 장치 /dev/rdsk/c0t3d0s2는 어느 구획을 표현하는가? 그리고 무엇을 의미하는가?
4. Linux에서 첫 구동기로서의 IDE하드디스크전체의 장치이름은 무엇인가? 만일 여기에 하나의 확장구획과 2개의 론리구획이 있다면 론리구획들의 장치이름은 무엇으로 되는가?
5. 하나의 주콤퓨터안에 있는 두개 파일이 같은 색인마디번호를 가질수 있는가?
6. 등록부는 무엇을 보관하는가?
7. 색인마디와 등록부안에서 공통적인 구성요소는 무엇인가?
8. UNIX파일이 파일끝(end-of-file)표식을 가지지 않는다. 이때 핵심부는 파일을 현시하는 경우 그 파일의 끝에 도달했다는것을 어떻게 아는가?
9. 상위블로크 및 색인마디와의 관계에서 sync의 역할은 무엇인가? 왜 체계를 끌 때 보통 2개의 sync가 요구되는가?
10. 구획표는 어디에 보관되는가? Linux에서 그것은 무엇으로 알려 지는가?
11. 교체파일체계를 가지는것이 왜 필요한가?
12. proc 또는 procfs파일체계의 3가지 중요한 특징의 이름을 부르시오.
13. SVR4에 의해 사용되는 표준파일체계는 무엇인가? 그의 특별한 특징은 무엇인가?
14. CD-ROM은 어느 파일체계를 사용하는가? 여기서 Rock Ridge의 역할은 무엇인가?
15. UNIX체계들은 상위블로크의 손상을 막기 위하여 어떻게 하는가?
16. 태우기등록부가 비여 있지 않다면 그우에 파일체계를 태울수 있는가?
17. mount는 파일체계들을 태우기 시작할 때 어느 파일을 읽어 들이는가?
18. Linux기계가 DOS나 Windows구획상의 거의 모든 파일이름에서 ~(물결표)를 보여 준다면 이것은 무슨 오유를 범한것인가? 긴 파일이름을 반대로 얻자면 어떻게 해야 하는가?
19. 파일체계가 읽기전용방식으로 태우기되였다면 그것을 읽기쓰기방식으로 태우기전에 먼저 내리워야 하는가?
20. 상위블로크의 기억기사본이 항상 디스크사본보다 더 최근의것인가?

제 22 장. 체계관리자의 일반임무

이 장에서는 일반적인 체계관리에 대하여 보기로 한다. 이 과제는 보통 한 사람 즉 **체계관리자**(system administrator)에게 위임된다. 체계관리자를 **상급사용자**(super user) 또는 **뿌리사용자**(root user)라고도 한다. 체계관리자는 상당한 권한을 가지고 있으며 실제적으로 모든것에 접근할수 있다. UNIX설치의 성공과 안정성은 넓은 범위에서 체계관리자의 능력에 달려 있다.

체계관리작업에는 보안을 유지하는것으로부터 시작하여 여벌복사를 해두고 디스크공간을 관리하며 사용자등록자리(account)를 유지하는것에 이르기까지 모든 체계관리가 포함된다. 스크립트들은 규칙적으로 수행되여야 할 조작들을 자동화하기 위하여 고안되여야 한다. 또한 봉사들은 체계의 구성파일들을 실행 및 편집하는것으로써 시작 또는 정지되여야 한다. 이 모든것들은 체계관리자가 체계의 각이한 구성요소들에 대한 깊은 지식을 가질것을 요구한다. UNIX는 대부분의 체계들보다 더 쉽게 관리되며 문서화가 잘되여 있으므로 그 부담은 그리 큰것이 아니다.

오늘날 모든 사용자들은 호출로부터 수행에 이르는 중요한 기능들의 일부는 알아야 한다. 이를 위해서는 일부 고급한 려파기들에 대한 지식을 가져야 하며 쉘프로그람작성에도 아주 능숙해야 한다. 지어 권한이 없는 사용자라고 해도 이 장의 대부분은 읽어야 한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 뿌리에 가입하여 su를 써서 상급사용자가 된다(22.1).
- passwd를 써서 사용자의 통과암호를 바꾼다(22.2.1).
- date로 체계날자를 바꾸거나 wall과 calendar를 써서 모든 사용자들과 통신한다 (22.2, 22.3).
- useradd, usermod, userdel을 리용하여 사용자등록자리를 만들고 수정 및 삭제한다(22.3).
- 사용자ID설정비트를 설정하여 프로그람들을 뿌리의 권한으로 실행킨다(22.4.1).
- 점착비트를 리용하여 등록부를 공유할수 있게 한다(22.4.2).
- 제한쉘을 리용하여 사용자의 활동을 제한한다(22.4.3).
- 체계기동과 끄기, 체계실행준위조종에서 init와 /etc/initab가 노는 역할을 배운다(22.5, 22.6).
- format, fdformat, dd를 리용하여 디스크를 초기화하고 복사한다(22.7.1, 22.7.2).
- DOS지령묶음을 리용하여 디스크상의 DOS파일들을 관리한다(22.7.3).
- cpio를 리용하여 파일들을 여벌복사하고 회복한다(22.8).
- tar를 리용하여 등록부나무를 여벌복사하고 파일들을 보존파일(archive)에 추가한다(22.9).
- du, find, xargs을 리용하여 디스크공간을 관리한다(22.10).
- passwd로 통과암호의 로화를 설정하여 통과암호가 규칙적으로 바뀌도록 한다(22.11).
- 체계데몬을 시동시키고 정지시키기 위해 init가 rc스크립트를 어떻게 사용하는가를 리해한다(22.12).
- getty가 어떻게 실행되며 말단에 대한 파라메터를 어떻게 설정하는가를 배운다(22.13).
- System V의 lp보조체계를 써서 인쇄준비, 인쇄기관리, 인쇄완충기의 조종을 배운다(22.14, 22.15).
- Linux에서 쓰이는 lpd체계를 리용하여 우와 같은 과제를 수행한다(22.15).

22.1 체계관리자의 가입(root)

UNIX체계는 관리자의 특정한 사용을 위하여 특별한 가입이름을 제공한다. 이러한 이름을 **뿌리**(root)라고 부른다. 이 등록자리(account)는 따로 만들 필요는 없고 모든 체계들에 존재한다. 그의 통과암호는 보통 체계설치시에 설정되며 가입할 때 사용된다.

```
login: root
password: *******[Enter]
# _
```

뿌리의 프롬프트는 권한을 가지지 않는 사용자들이 리용하는 $나 %와 달리 #이다. 일단 뿌리로 가입하면 뿌리의 홈등록부에 배치된다. 체계에 따라 이 등록부는 /이거나 /root일수 있다.

현대체계들에서 대부분의 관리지령들은 /sbin과 /usr/sbin에 존재한다. 그러나 낡은 체계를 사용하고 있다면 그것들을 /etc에서 찾아야 한다. 뿌리의 PATH목록도 다른 사용자들과 다르다.

```
/sbin:/bin:/usr/sbin:/usr/bin:/usr/dt/bin
```

상급사용자는 아무때나 파일체계안의 임의의 위치로 움직일수 있는것으로 하여 다른 사용자들에 의해 작성된 스크립트들과 프로그람들을 무심결에 실행시킬수 있다. 이것이 상급사용자를 위한 PATH변수가 현재의 등록부를 포함하지 않는 리유이다.

주해

체계관리작업을 위하여 UNIX체계들에 지원되는 많은 표준스크립트(특히 기동과 관계되는 스크립트)들은 Bourne쉘에서 실행한다. Korn쉘을 사용할 때는 거기서 개발된 스크립트들이 다른 주콤퓨터(Korn쉘을 가지고 있지 않는)에서 실행되지 않으므로 심중해야 한다. 더우기 C쉘스크립트를 사용하는 환경하에 있지 말아야 한다. Linux는 일반관리활동과 체계관리활동의 두 경우에 다 bash를 사용한다. 이 경우에는 우와 같은 문제가 생기지 않는다.

상급사용자상태얻기(su)

뿌리통과암호를 알고 있는 사용자는 su지령을 써서 상급사용자상태(super user status)를 얻을수 있다. 실례로 사용자 local(홈등록부 /home/local을 가진)이 상급사용자로도 될수 있다.

```
$ su
Password: ********[Enter]          뿌리의 통과암호이여야 한다
# pwd                    프롬프트는 변경되지만 등록부는 변경되지 않는다
/home/local
```

여기서 su는 뿌리의 통과암호를 요구한다. 프롬프트 #는 사용자가 상급사용자이라는것을 가리킨다. 사용자ID($LOGNAME)는 여전히 국부적이며 현재등록부는 변화되지 않는다. 사용자는 이제부터는 상급사용자의 권한을 가진다.

사용자환경의 만들기

사용자들은 자주 프로그람의 실행이 중단되는 불만을 관리자에게 제기하군 한다. 관리자는 먼저 모의환경에서 그것을 실행시켜 본다. su는 -와 함께 사용되였을 때에는 가입-통과암호라는 경로(route)를 취하지 않고 사용자의 환경을 다시 만들어 준다.

```
su - henry               통과암호를 요구하지 않는다
```

우의 지령은 henry의 시작파일(.profile 또는 임의의 다른 파일)을 실행시키고 관리자에게 henry의 환경을 만들어 준다. su가 보조쉘에서 실행되므로 이 방식은 [Ctrl-d]를 누르거나 exit지령을 사용하여 끝낼수 있다. 다른 사용자의 등록자리로 가입하지 않고도 상급사용자는 -c선택항목을 리용하여 그 환경에서 프로그람을 실행시킬수 있다. 실례로 지령 dbstart가 Oracle사용자등록자리에 의하여 실행되여도 뿌리에서 이 지령으로 Oracle자료기지를 시동시킬수 있다.

```
su oracle -c dbstart
```

체계의 시동스크립트안에 이러한 지령들을 배치하여 자료기지는 체계기동시에 자동적으로 시동되게 할수도 있다.

22.2 관리자의 권한

상급사용자는 막대한 능력을 가지며 그의 특정한 사용을 위해 예약된 몇개의 지령들이 있다. 일부 지령들도 관리자에 의해 실행될 때에는 좀 다르게 동작한다. 상급사용자의 권위는 주로 다음의 권한으로부터 나온다.

- 어떤 파일의 속성 즉 그의 내용이나 허가권 지어는 소유권를 변화시킨다. 그는 어떤 파일이 쓰기 보호되여 있다고 할지라도 rm을 리용하여 그것을 삭제할수 있다.
- 임의의 프로쎄스를 초기화 또는 제거한다. 관리자는 체계를 실행시키는데 필용한것들을 제외한 모든 프로쎄스들을 직접 제거할수 있다. 상급사용자만이 체계를 끄기 위해서 shutdown지령을 사용할수 있다.
- 다른 사용자의 통과암호를 모르고도 그것을 변화시킨다.
- 체계시간을 설정한다.
- 모든 사용자들에게 동시에 주소를 달아 준다.
- 사용자들이 만들수 있는 파일들의 최대크기를 제한한다.
- at나 cron과 같은 일정작성봉사(scheduling service)들에 대한 사용자의 접근을 조종한다.
- TCP/IP봉사인 rlogin, rcp, rsh에 대한 사용자접근을 조종한다.

관리자는 이 권한들을 최대한의 주의를 가지고 사용해야 한다. 보기에 흠이 없고 장애물이 없는것 같은 빈틈도 그에 대한 지식을 장난기 있는 사람들이 얻는 경우에는 큰 재난을 일으킬수 있다. 일정관리(scheduling)와 TCP/IP봉사들에 대한 사용자접근은 이미 취급하였으므로 앞으로 다음절들에서 나머지 내용들을 설명한다.

22.2.1 통과암호변경(passwd)

passwd는 권한 없는 사용자에 의해 사용될 때에는 현재의 통과암호를 문의한다. 그렇지만 상급사용자가 이 지령을 사용하면 체계는 더 관대한 방식으로 동작한다.

```
# passwd
changing password for root
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
```

```
New password: ********
Re-enter password: ********            반복하여 입력한다
Password changed.
```

체계가 이때 낡은 통과암호를 문의하지 않는다는것을 주의해야 한다. 관리자는 상급사용자통과암호를 철저히 지켜야 할뿐아니라 그것을 기억해야 한다. 만약 그렇지 않으면 전체 UNIX체계를 재적재해야 할수도 있다. UNIX는 또한 관리자에게 다른 사람의 통과암호를 변화(그것을 모르는 상태에서)시킬수 있는 권한을 준다.

```
passwd henry
```

낡은 통과암호는 문의되지 않으며 오직 새 통과암호를 2번 입력해야 한다.

최근에는 체계내부로 억지로 들어 와 보려고 하는 해커들이 있다. 사용자들은 흔히 자기의 통과암호를 다른 사람에게 내주며 그때문에 체계의 안전이 위태롭게 된다. 결과적으로 통과암호들은 시간이 감에 따라 다른 사람들에게 알려 지는 경향이 있다. 문제를 더 위태롭게 만드는것은 사용자들자신이 자기의 통과암호를 바꾸는것을 아주 싫어 한다는것이다. passwd지령은 사용자들에게 제정한 시간후에 자기의 통과암호를 바꿀것을 요구하도록 하는 기능을 제공한다. 이 기능에 대해서는 22.11에서 설명한다.

주해

passwd는 상급사용자가 사용하였을 때에는 뿌리통과암호를 변화시키는 경우에조차도 낡은 통과암호를 묻지 않는다.

22.2.2 체계날자의 설정(date)

우리는 앞에서(3.9) 체계날자를 현시하기 위하여 피동적인 지령으로써 date를 사용하였다. 관리자의 손에서는 이 지령이 실제적으로 체계날자를 설정하기 위한 수값인수와 함께 사용될수 있다. 이 인수는 보통 MMDDhhmm의 형식을 가진 8문자길이의 문자렬이며 년문자렬을 2자리 또는 4자리로 선택할수 있다.

```
# date 06010735
Fri Jun  1 07:35:00 2000
```

UNIX체계들은 일정한 기간(적어도 2038년까지)은 세기를 리해할것이다. 일정작성프로그람 cron은 일감을 실행시키기 위해 박자시간을 사용하므로 날자가 정확한가를 확인해야 한다.

주해

체계날자는 오직 관리자만이 설정할수 있다.

22.2.3 사용자들과의 통신(wall과 calendar)

wall지령은 모든 사용자들에게 동시에 주소를 달아 준다. 거의 모든 UNIX체계들은 사용자들에게 이 지령의 사용을 허가하지 않으며(Linux제외) 관리자만이 쓸수 있게 예약되여 있다.

```
# wall
The machine will be shut doun today
at 14:30 hrs. The backup will be at 13:30 hrs
[Ctrl-d]
```

현재 가입되여 있는 모든 사용자들이 자기의 말단에서 이 통보문을 받는다. mesq설정은 wall지령에 의해서 무시된다. 이 지령은 관리자에 의하여 일상적으로 실행되며 특히 체계를 닫기전에 실행된다.

우리는 이미 유용한 재생봉사(reminder service)로서 calendar를 사용해 보았다. 관리자로서는 모든 사용자들에게 자기들의 약속을 상기시키기 위해 인수(-)와 함께 사용할수 있다.

```
calendar -
```

calendar는 모든 사용자들의 calendar파일을 읽고 사용자들에게 우편을 보낸다. 그 지령은 관리자의 프로필(/.profile) 또는 기동시에 실행되는 시작스크립트안에 배치하면 좋다. 더 좋기는 매일 지정된 시간에 cron이 그것을 실행하게 하는것이다.

22.2.4 파일크기의 한계설정(ulimit)

결함 있는 프로그람이나 프로쎄스들은 아무때나 디스크공간을 차지할수 있다. ulimit지령은 사용자가 만들수 있는 최대크기를 제한한다. ulimit가 홀로 사용되였을 때에는 현재의 설정을 현시한다.

```
# ulimit
2097151                Linux와 Solaris는 통보문 "unlimited"를 보여준다
```

512(일부 체계들에서는 1024)byte단위로 표현되는 기정한계는 핵심부안에 설정된다. 보통의 사용자는 이 초기값을 감소시킬수만 있으나 상급사용자는 그것을 증가시킬수도 있다.

```
ulimit 20971510                10배 증가된 최대화일크
```

흔히 체계관리자들은 이 명령문을 /etc/profile에 넣어 모든 사용자들이 이 제한밑에서 작업하도록 한다. 현재 이 지령은 모든 쉘들의 내장(built-in)지령이며 이 책에서 설명되지 않는 많은 선택항목들을 제공한다.

22.3 사용자등록자리의 관리

UNIX에서 사용자라는 말은 오직 사람만을 의미하지 않는다. 더우기 그것은 대상과제(project)나 응용프로그람을 표현할수 있다. 류사한 기능을 가지는 사용자들의 그룹는 같은 사용자이름으로써 체계를 사용할수 있다. 그러므로 marketing, accounts, mis 등과 같은 사용자이름들을 가지는것이 아주 일반적이다. 사용자등록자리들의 만들기와 유지를 위하여 UNIX는 3개의 지령 즉 useradd, usermod, userdel을 제공한다.

사용자등록자리를 열 때 관리자는 그 사용자를 어떤 그룹에 련결시켜 주어야 한다. 그룹은 보통 서로 다른 권한설정을 가지는 한명이상의 성원들을 가진다. 공통대상과제에 봉사하는 사람들은 다른 사람의 파일들을 읽을수 있으며 그것은 그들이 같은 그룹에 소속될 때에만 가능하다. chmod와 chgrp지령들은 그룹에 적용하는 파일속성들을 변화시킬수 있다.

사용자는 다음과 같은 파라메터설정을 할수 있다.

- 사용자식별번호(UID)와 사용자이름
- 그룹식별번호(GUID)와 그룹이름
- 홈등록부
- 가입쉘

- /var/mail안의 우편함
- 통과암호

/etc/passwd파일을 보면 이 파라메터들이 대체로 매 사용자에 해당한 행에 들어 있다. 초기에는 체계관리자가 홈등록부와 우편함을 수동으로 만들어야 하였지만 오늘날에는 이 모든 작업을 수행하는 하나의 지령이 있다. 이제 어떤 사용자를 위한 그룹을 만들고 그 사용자를 체계에 추가하여 보자.

22.3.1 그룹의 추가(groupadd)

사용자가 새로운 그룹에 배치되려고 한다면 먼저 그 그룹을 위한 기입항목을 /etc/group안에 만들어야 한다. 사용자는 항상 하나의 1차그룹(primary group)을 가지며 하나이상의 2차그룹(secondary group)을 가질수도 있다. 이 파일에는 체계의 모든 그룹들이 들어 있으며 이 파일의 몇개 행들을 보면 그 구조를 알수 있다.

```
root:x:0:root
bin:x:1:root,bin,daemon
lp:x:7:
uucp:x:14:uucp,fax,root,fnet,sumit
users:x:100:henry,oracle,image,enquiry
pppusers:x:230:                GUID는 230이다
```

매행은 4개의 마당으로 되여 있다. 여기서 users라는 그룹의 기입항목을 고찰하자.

- 첫번째 마당(users)은 그룹이름이다. 이것은 목록의 그룹소유권렬에서 본것과 같은 이름이다.
- 두번째 마당은 비여 있지 않으면 x가 들어 있다. 이전에는 여기에 그룹통과암호가 들어 있었지만 지금은 일반적으로 사용되지 않는다.
- 셋번째 마당은 그 사용자의 수값화된 그룹식별번호(GUID)이다(여기서는 100). 같은 그룹에 속하는 두 사용자는 같은 GUID를 가진다.
- 마지막마당에는 반점으로 구분된 사용자이름들의 목록(henry, oracle, image, enquiry)이 들어 있다. 이것들은 추가적인 사용자들이며 2차그룹에 속하게 된다. 이 마당이 비여 있다고 해서 그룹에 성원이 없다는것을 의미하지는 않는다.

241의 GUID를 가진 새로운 그룹 dba를 만들기 위해서는 groupadd지령을 사용해야 한다.

```
groupadd -g 241 dba
```

우의 지령은 /etc/group 안에 기입항목을 배치하는데 다음과 같이 직접 삽입할수도 있다.

```
dba:x:241:
```

일단 그 그룹을 위한 기입항목(entry)이 만들어 지면 체계에 사용자를(그룹과 함께) 추가할수 있게 된다.

주해

groupadd와 별도로 그룹을 수정하고 삭제하기 위하여 groupmod와 groupdel지령을 사용할수도 있다.

22.3.2 사용자의 추가(useradd)

useradd지령은 체계에 새로운 사용자들을 추가한다. 지령행에 사용자와 관련되는 모든 파라메터들이 제공해야 한다.

```
# useradd -u 210 -g dba -c "THE RDBMS" -d /home/oracle -s /bin/ksh -m oracle
# _
```

이것은 210의 UID와 그룹이름 dba를 가진 사용자 oracle을 간단히 생성한다. 홈등록부는 /home/oracle이며 사용자는 Korn쉘을 사용한다. -m선택항목은 홈등록부가 없다면 그것을 만들고 사용자의 표본 .profile과 .kshrc를 복사해 준다. useradd가 /etc/passwd안에 만든 행을 그림 22-1에서 보여 준다.

그다음은 지령 passwd oracle을 써서 새 사용자의 통과암호를 설정해야 한다. 일단 이 모든것이 수행되면 oracle사용자등록자리가 사용될수 있다.

22.3.3 사용자프로필 /etc/passwd와 /etc/shadow

통과암호부호화(password encryption)를 제외한 모든 사용자정보들은 /etc/passwd안에 보관된다. 이전에는 이 파일에 통과암호가 들어 있었기때문에 공개되여 있었다. 부호화(encryption) 그자체는 /etc/shadow안에 기억된다. 이 파일이 현재 사용자통과암호의 합법성을 확정하기 위해 passwd지령이 사용하는 조종파일이다.

/etc/passwd안에서 oracle에 해당한 행을 얻어 내자. 여기에는 7개의 마당이 있으며 그 의미를 보기로 하자(/etc/passwd에 반영된 순서로).

- 사용자이름 - UNIX체계에 가입할 때 사용하는 이름(oracle).
- 통과암호 - 여기서는 통과암호부호화를 나타내지 않으며 대신 x가 들어 있다.
- UID - 사용자의 수값형식의 식별자(210). 사용자마다 UID가 서로 다르다.
- GUID - 사용자의 수값형식의 그룹식별자(241). 이 수는 /etc/group의 3번째 마당에도 있다. ls와 같은 지령들은 그룹이름을 출력하기 위해 이 파일을 읽어야 한다.
- 설명문 또는 GCOS - 사용자의 세부정보 즉 이름, 주소 등(The RDBMS). 이 이름은 전자우편주소의 앞에 사용된다. 이 사용자등록자리로부터 발송된 우편은 송신자 "The RDBMS" <oracle@planets. com>으로 표현할것이다(사용자가 그 령역에 속한다고 가정한다).
- 홈등록부 - 사용자가 가입을 끝내는 등록부(/home/oracle).
- 가입쉘 - 가입한후 실행되는 첫 프로그람. 보통 쉘이다(/bin/ksh).

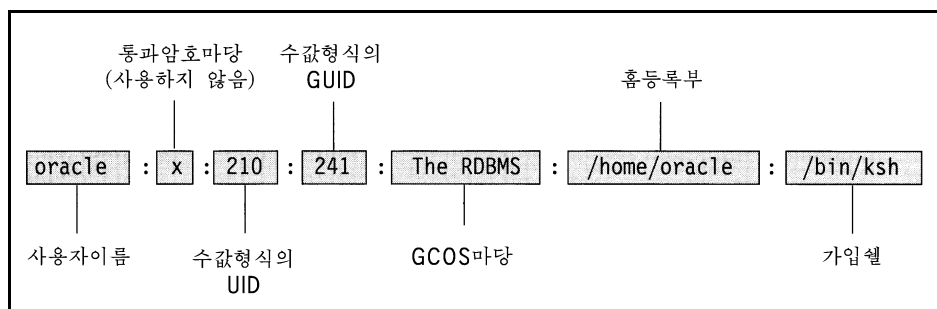



그림 22-1. /etc/passwd의 행구조

이 파일은 보통 관리자에 의해서만 변경될수 있지만 사용자들은 관리자와의 교제가 없이 chsh지령을 써서 자기들의 쉘을 변화시킬수 있다(17.1).

주해

/etc/passwd안의 마지막마당은 사실상 사용자가 가입할 때 실행되여야 하는 지령이다. 이것은 보통 쉘이지만 관리자는 사용자의 활동을 제한하기 위해 다른 프로그람(pppd와 같은)을 선택할수도 있다.

/etc/passwd의 모든 행에 대하여 /etc/shadow안에 대응하는 항목이 있다. 이 파일안의 관련행들은 아래와 같다.

```
oracle:$1$07VbeHwq$0qHK0W73boShhr093.txp.:10846:-1:99999:-1:-1:-1:135365660
```

통과암호부호화는 두번째 마당에 나타난다. 이 부호화로부터 통과암호를 만들어 내는것은 불가능하다. 그렇지만 솜씨 있는 해커들은 부호화알고리듬(encryption algorithm)을 사용하여 부호화된 패턴의 렬을 발생시킬수 있다. 그러한 알고리듬들은 인터네트상에서도 광범하게 리용되고 있다. 해커들이 본래의 암호를 찾아 내는것은 충분히 가능하며 그러므로 이 파일은 상급사용자가 아닌 다른 모든 사용자에게는 읽혀 질수 없게 만들어야 한다.

22.3.4 사용자의 수정과 삭제(usermod, userdel)

usermod는 useradd로 설정한 파라메터의 일부를 수정하기 위해 사용된다. 사용자들은 때때로 자기의 가입쉘을 변화시킬 필요가 있다. 다음의 지령행은 사용자 oracle을 위한 가입쉘로써 bash를 설정한다.

```
usermod -s /bin/bash oracle
```

userdel을 리용하여 사용자들을 체계에서 삭제할수 있다. 다음의 지령은 사용자 oracle을 체계에서 삭제한다.

```
userdel oracle
```

이것은 /etc/passwd, /etc/group, /etc/shadow로부터 oracle에 해당한 모든 항목을 제거한다. 이때 사용자의 홈등록부는 삭제되지 않으며 필요하다면 개별적으로 삭제해야 한다.

22.4 보안관리

콤퓨터체계안에서 보안은 파일들과 관계되기때문에 불결한 파일허가권은 악의 있는 사용자들에 의하여 파괴적인 방법으로 쉽게 침해 당할수 있다. 관리자는 체계등록부(/bin, /usr/bin, /etc, /sbin 등)들과 그안의 파일들이 다른 사람에 의해 변경될수 없도록 해야 한다. 이제부터 UNIX체계에서 볼수 있는 여러가지 보안관련구조의 일부를 설명한다.

22.4.1 림시적인 권한(사용자ID설정방식)

많은 UNIX프로그람들은 /etc/shadow와 같이 사용자들이 직접 편집기로 수정할수 없는 예민한 체계파일들을 갱신할수 있게(사용자들이) 하여 준다. 그것은 프로그람들이 실행중에 사용자들이 파일소유자의 권한을 얻을수 있게 하여 주는 특수한 허가권방식을 가지기때문이다. 아래에 실제의 passwd프로그람을 주었다.

```
-rwsr-xr-x   1  root    shadow    34808 Nov 30 17:55 /usr/bin/passwd
```

허가권마당의 문자 s는 **사용자ID설정방식**(set-user-id:SUID)이라고 하는 특수한 방식이다. 권한 없는 사용자가 passwd를 실행할 때 그 프로쎄스의 효과적인 UID는 사용자의것이 아니고 뿌리의것 즉 그 프로그람소유자의것이다. 이 SUID권한을 리용하여 passwd가 /etc/shadow를 편집한다. 이 권한은 그 프로그람의 완료와 함께 없어 진다.

파일의 SUID는 오직 상급사용자만이 chmod지령의 특수한 문법을 써서 설정할수 있다.

```
# chmod u+s filex ; ls -l filex
-rws-x-x   1 root   bin    113 Mar 24 11:18 filex
```

SUID는 보안위험요소이다. 일단 어떤 사용자가 뿌리가 소유한 그러한 파일에 접근하였다면 그는 은폐된 능력을 얻게 된다(지어 그는 그것을 알아 채지 못할수도 있다). 관리자는 사용자가 만들거나 복사해 볼수도 있는 뿌리소유의 모든 SUID프로그람들의 경로를 알고 있어야 한다. find지령이 그것들의 위치를 쉽게 찾아 낸다.

```
find /home -perm -4000 -print | mail root
```

find에서 추가적인 8진비트(4)는 SUID방식을 표시하며 000은 임의의 다른 허가권을 표시한다. cron을 사용하여 규칙적인 시간간격으로 이 프로그람을 실행시켜 파일목록을 뿌리에 우편으로 보낼수 있다.

데니스 리치에(Dennis Ritchie)에 의하여 발명된 SUID기구는 UNIX에만 있는 특징적인 부분이다. **그룹ID설정방식**(set-group-id:SGID)은 그 ID가 설정된 프로그람이 사용자로 하여금 그 프로그람을 소유한 그룹과 꼭 같은 능력을 가지도록 한다는것을 제외하고는 사용자ID설정방식과 류사하다. SGID는 2번째 비트이다.

주해

4번째 비트는 오직 파일의 특수한 방식들이 설정될 때만 사용된다. 그것은 SUID에 대해서는 4, SGID에 대해서는 2, 점착비트에 대해서는 1의 값을 가진다. 다른 3비트들은 일반적인 의미를 가진다.

22.4.2 점착비트

《무겁게》읽혀 지는 체계상의 프로그람들은 디스크의 교체구역에로 옮겨 지며 필요할 때 다시 읽혀 진다. **점착비트**(sticky bit)는 교체가 한번만 진행되며 교체구역에 그 프로그람의 본문자료(10.2)를 영구적으로 기억하게 해준다. 허가권마당의 문자 t(4번째 8진비트 1)가 그것을 가리킨다.

```
-rwxr-xr-t    2 root     root     2878448 Sep 25 1999 /usr/bin/emacs
```

이것은 emacs가 일단 사용되였다면 그것이 기계가 닫길 때까지 교체구역에 남아 있게 된다는것을 의미한다. 따라서 하드웨어상에서 편집기가 처음으로 시작할 때는 일정한 시간이 걸리지만 다음번부터는 그렇지 않다. 그렇지만 그것은 디스크구동기속도가 느리고 RAM용량이 부족한 기계들에서나 나타나는것이므로 이제는 이 비트가 자기의 거의 모든 매력을 잃어 버렸다. 초고속의 디스크구동기와 대량의 값 눅은 기억기를 가진 현대적인 기계들인 경우 일반파일들에 대해서는 점착비트가 필요 없다.

등록부에 점착비트를 사용

그러나 점착비트를 등록부에서 사용할 때에는 쓸모 있는 보안구조로 된다. UNIX체계에서 사용자들은 /tmp와 /var/tmp안에 파일들을 만들수 있지만 누구도 자기에게 소유되지 않은 파일들을 지울수는 없다. 그것은 두 등록부가 다 자기의 점착비트를 설정하고 있기때문이다.

```
# ls -l /tmp /var/tmp
drwxrwxrwt  15 root    root      6144 Nov 28 22:26 /tmp
drwxrwxrwt  37 root    root      1024 Nov 27 23:46 /var/tmp
```

모두가 그 등록부들에 써넣을수는 있지만 추가적인 t비트는 henry가 이 등록부안에 있는 romeo의 파일을 삭제할수 없게 한다. chmod를 사용하여 추가적인 비트로 1을 써서 등록부상에 그 비트를 설정할수 있다.

```
# chmod 1775 bar
# ls -l bar
drwxrwxr-t   2 sumit    dialout     1024 Apr 13 08:25 bar
```

점착비트는 그룹대상과제를 실현하는데서 아주 유용하다. 한 그룹의 사용자들이 보안을 위반하지 않고 파일묶음을 가지고 작업하도록 하려면 다음과 같이 해야 한다.

- 이 사용자들을 위한 공통그룹을 /etc/group안에 만든다.
- 그것들에 대한 사용자등록자리들은 따로 만들지만 같은 홈등록부를 지정한다.
- 그 홈등록부와 모든 보조등록부들이 어느 사용자에게만 소유되지 않도록 확인한다. 소유권을 뿌리에로 넘겨 주기 위해서는 chown을 사용한다.
- 그 등록부들을 그룹적으로 또는 일반적으로 쓰기가능하게 만들며 chmod 1775를 써서 점착비트를 설정한다.

이 실례에서 그룹의 모든 사용자는 그 등록부들상에서 쓰기허가권을 가지며 파일들과 등록부들을 만들수는 있으나 자기가 소유한것들만 지울수 있다. 이것은 사실상 아주 쓸모 있는 특징이다.

참고

한 그룹에 속하는 사용자들이 공유하는 등록부를 만들려면 점착비트를 설정한다. 매 사용자가 소유한 파일들은 그 그룹의 다른 사용자들의 간섭으로부터 보호된다. 그 등록부는 소유되지 말아야 한다.

22.4.3 제한쉘

사용자의 활동을 제한하기 위해서는 특별히 제한쉘(restricted shell)을 가지고 사용자등록자리를 설정해야 한다. 이 쉘은 rsh라는 이름을 가지고 있었지만 오늘날 rsh는 프로그람을 원격으로 실행시키는 지령으로 쓴다. 지금은 rbash와 rksh와 같은 더 좋은 제한쉘들이 있다. 그것들중 어느것이든지 /etc/passwd의 마지막마당에 지정되여야 한다. 제한쉘을 가진 사용자는 다음과 같은것들을 할수 없다.

- cd지령을 사용하는것. 즉 등록부를 바꿀수 없다.
- PATH를 재정의하는것. 이것은 다른 등록부에 배치된 지령들에 접근할수 없게 한다.
- SHELL을 재정의하는것. 따라서 사용자는 비제한쉘로 교체할수 없다.
- /를 포함하는 경로이름을 사용하는것. 즉 지령이 상대 및 절대경로이름으로 실행될수 없다.
- 파일만들기 및 추가를 위한 >와 >>연산자를 사용하는것

이러한 환경에서 사용자는 새롭게 변경시킬수 없는 PATH에서 지정된 등록부의 프로그람들만을 실행시킬수 있다. 이것은 보통 현재등록부만으로 설정된다. 만일 사용자가 /bin과 /usr/bin안에 있는 일부 체계지령들을 실행시킬 필요가 있다면 그 사용자의 제한등록부(restricted directory)안에 그 지령의 련결을 만들어야 한다.

일부 지령들은 쉘탈퇴(vi나 mail과 같이)들을 가지며 UNIX의 일부 판본들은 이 탈퇴들을 사용하여 절대경로이름으로 임의의 UNIX지령을 실행시키게 한다. 이 지령들이 체계안에서 그러한 방법으로 동작하지 않는가를 확인해야 한다. 만일 그렇게 동작한다면 그 사용을 금지시켜야 한다.

참고

자기의 체계상에 제한쉘을 따로 가지고 있지 않다면 그러한 동작을 시키도록 -r선택항목을 가지고 표준쉘을 사용할수 있다(sh -r, bash -r, ksh -r). 이 항목들은 /etc/passwd안에 넣을수 없으므로 보통 쉘을 실행시키고 exec를 써서 시작파일로부터 그것들을 실행시킬수 있다. PATH가 하나의 등록부로 설정되도록 해야 한다.

22.5 체계기동

시동 및 닫기절차는 아주 드물게 변화되는 자동화된 쉘스크립트에 의해 조종된다. 관리자는 시동 및 닫기를 진행하는 동안 체계가 따라 가는 단계들의 정확한 순서를 알고 있어야 한다. 특히 시작할 때 오유가 생길수 있으며 관리자는 그것들을 해결할수 있어야 한다.

22.5.1 첫 이동자(init)

체계시동시에 초기화되는 몇개의 프로쎄스들이 있다. 핵심부(/stand/unix, /kerne/genuix 또는 /vmlinuz)가 주기억에 읽혀 지며 다음프로쎄스들이 뒤따라 생성되기 시작한다. 이중에서 가장 중요한것은 PID 1을 가진 /sbin/init이며 이것은 뒤따르는 모든 프로쎄스들의 생성에 대하여 책임을 진다. 2개의 중요한 리유로 하여 init의 동작패턴을 알아야 한다.

- 실행준위(체계상태)를 조종하며 매 실행준위를 위하여 어느 프로쎄스를 실행시키겠는가(또는 제거하겠는가)를 결정한다.
- 모든 말단 및 모뎀포구에 getty프로쎄스를 생성하여 사용자들이 가입할수 있게 한다.

init는 또한 모든 체계데몬들이 실행하고 있는가도 확인한다. lpsched는 인쇄를 위하여 대기하고 있는 일감들을 위한 행인쇄완충기(line printer spooler)를 관리한다. cron은 체계의 크로노그라프(chronograph:시간을 도형적으로 기록하는 장치)이다. httpd는 Web봉사기데몬이며 sendmail은 들어오고 나가는 모든 우편들을 감시한다. init는 그것들모두의 어미(때때로 어미웃준위)로 존재한다.

22.5.2 실행준위(init)

init는 체계가 **실행준위**(run level)라고 하는 여러가지 상태에서 유지되도록 한다. 여기서 매 실행준위는 보통 한자리수자(0-6) 또는 s나 S이다. 이 매 상태에서 실행될 프로쎄스들의 특정한 묶음이 미리 정해 져 있다. 보통 체계는 이 실행준위의 어느 하나에 있게 된다.

0 - 체계닫기

1 - 체계관리방식(국부파일체계들이 태워 진다.)

2 - 다중사용자방식(NFS는 리용할수 없다.)

3 - 완전한 다중사용자방식

6 - 닫기 및 재기동방식

s 또는 S - 단일사용자방식(파일체계들이 태워 진다.)

우리는 실행준위 4와 5가 우리와 관계 없는 곳에서 사용되든 사용되지 않든지간에 그것들은 고찰하

지 않겠다. 체계가 기동될 때 init는 먼저 실행준위 1(체계관리자방식)로 들어 간다. 체계데몬들이 실행되지 않고 있기때문에 인쇄기나 말단들은 사용할수 없다.

단일사용자방식은 관리자가 비직결여벌복사(offline backup)를 하는것과 같은 관리일감을 수행하기 위해 사용된다. 실행준위 1의 역할은 체계에 따라 다르다. 즉 일부 체계들에서는 실행준위 1과 S가 동일한 역할을 수행한다.

체계에 따라 표준적인 다중사용자방식은 2개의 실행준위(2 혹은 3)중 어느 하나로 실현된다. 실행준위를 인수로 하여 init지령을 사용하면 실행준위를 바꿀수 있다.

```
init 2
init 3
```

기동통보문(혹은 /etc/inittab)을 주의 깊게 보면 체계의 표준적인 다중사용자실행준위를 찾아 낼수 있다.

현재의 실행준위를 알기 위해서는 who -r지령을 사용하시오. Linux에서는 runlevel지령을 사용하시오.

22.5.3 init의 시작파일 /etc/inittab

init의 동작은 /etc/inittab에 의하여 조종된다. init는 실행될 때 이 파일의 모든 기입항목을 읽는다. 그의 마당들은 init의 매 실행준위에 관하여 생성되여야 할 프로쎄스들과 통신포구들에서 실행될 프로쎄스들을 결정한다. 그림 22-2에 있는 몇개의 표본행들을 보자.

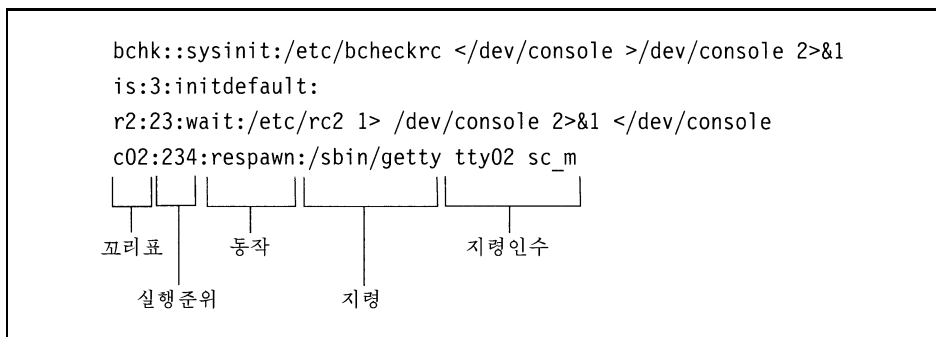



그림 22-2. /etc/inittab파일

시동시에 일어 나는 모든것들은 다음과 같은 항목에 최종적인 기원을 두고 있다. 표식(label)은 단순히 기입항목(entry)들을 구별하기 위해 사용되며 진짜 의미는 없다. 2번째 마당은 이 행이 적용될수 있는 실행준위를 보여 준다. 마지막 두개의 마당에는 동작(action)과 지령(command)이 있다(지령인수도 지령의 부분으로 본다면).

이제 inittab의 어느 한 행을 분석하여 보자. 표식 r2를 가진 행은 《실행준위 2 또는 3에 대하여 /etc/rc2프로그람을 실행하고 이 파일의 다른 행으로 옮겨 가기전에 그 프로그람이 완성되기를 기다리라.》라는 지시를 준다. 모든 입력, 출력 및 오유통보문들은 조종탁을 통해 지시되여야 한다. 다른 행들의 의미는 후에 보기로 하자.

init가 지정한 실행준위를 인수로 하여 실행될 때는 그 실행준위와 맞는 모든 행들을 읽고 거기에

지정된 지령들을 순서대로 실행시킨다. 실행준위가 없는 행(여기서는 첫번째 행)의 지령은 모든 실행준위에 대하여 실행되여야 한다. init는 또한 동작(action)으로써 initdefault를 보여 주는 행을 읽으면 초기의 실행준위를 얻는다. 여기서 체계는 실행준위 3으로 기동한다.

initdefault와 wait는 init가 리해하는 2개의 동작(action)이다. 이외에도 다음과 같은것들이 있다.

- sysinit – 체계를 초기화하는데 쓰인다. 체계는 파일체계의 불결정도를 검사하고 교체구획을 활성화하며 주콤퓨터이름을 설정할수도 있다. 또한 관리자로부터의 입력을 요구할수 있다.
- respawn – 완료시에 재시동하는 프로쎄스를 확인한다. 이것은 getty프로쎄스를 위해 항상 요구된다.
- once – 프로쎄스를 단 한번만 실행하며 그의 완성을 기다리지 않는다.
- boot – inittab가 처음으로 읽혀 질 때만 실행한다. init는 여기에 배치된 실행준위 마당을 무시한다.
- bootwait – 우와 같으나 그의 완성을 기다린다.
- off – 실행하고 있는 프로쎄스를 제거한다.
- ctrlaltdel – shutdown지령을 실행시킨다(Linux에서만).

telinit q의 사용

관리자들은 /etc/inittab안에 명령들을 삽입 또는 수정할수도 있다. 기정실행준위를 변화시킬수 있으며 또는 체계에 새로운 말단이나 모뎀이 추가될 때 기입항목을 추가, 변경할수 있다. 그러나 그때 init가 자기의 구성파일을 다시 읽도록 telinit지령을 사용해야 한다.

```
telinit q
```

init와 telinit는 련결되므로 telinit는 ln지령(존재하지 않는다면)을 써서 언제나 만들어 질수 있다. init q를 사용할수도 있다.

주해

/etc/inittab안에 initdefault를 포함하는 기입항목을 찾아 보면 기정실행준위를 얻을수 있다.

22.6 체계끄기

관리자는 하루의 작업이 끝나면 기계의 전원을 끌 의무도 가지고 있다. shutdown지령이 이 절차를 조종한다. shutdown은 보통 다음과 같은 동작을 수행한다.

- wall을 써서 체계에서 탈퇴하라는 지시와 함께 체계가 완료된다는것을 사용자들에게 통지한다. 사용자들에게 몇분동안에 자기의 모든 파일들을 닫고 체계에서 탈퇴할것을 요구한다. shutdown 자체는 첫 통보문을 보낸후에 몇분동안 기다리며 한두번 재촉을 요구할수도 있다.
- 모든 실행중의 프로쎄스들에 정상적으로 끝마칠수 있도록 신호를 보낸다.
- 사용자들을 체계에서 탈퇴시키고 남은 프로쎄스들을 제거한다.
- 모든 보조파일체계들을 내리운다.
- 파일체계의 완전성을 보호하기 위하여 디스크에 파일체계상태에 대한 정보를 써넣는다(2.5.4).
- 체계를 재기동하거나 차단 또는 단일사용자방식으로 전환한다는것을 사용자들에게 통보한다.

아래와 같은 통보문이 조종탁(console)에 나타나면 기계가 완료절차를 성과적으로 완성한것으로 볼수 있다.

```
Reboot the system now or turn power off
System halted
```

이제는 전원을 끄거나 체계를 재기동할수 있다.

-g선택항목은 1분간의 기정기다림시간을 재정의할수 있다. 그 지령은 아래와 같은 방법으로 사용될수 있다.

shutdown -g2	2분후에 기계의 전원을 끈다
shutdown -y -g0	즉시에 완료한다
shutdown -y -g0 -i6	완료하고 재기동한다(실행준위를)

init가 체계를 완료하기 위해 실행준위 0과 6을 사용하므로 init를 이러한 목적에 사용할수도 있다 그러나 shutdown에 비하여 좋지 못한 방법이다.

```
init 0
init 6
```

주해

Solaris와 같은 일부 체계들은 사용자들에게 경고함이 없이 체계를 완료할수 있는 reboot와 halt지령을 가지고 있다. 또 다른 지령 즉 haltsys는 체계를 즉시에 꺼버린다. 다중사용자체계를 관리하고 있는 경우에는 shutdown을 써야 한다.

Linux

[ctrl][Alt][Del]로도 체계를 끌수 있다. 모든 Linux체계들은 inittab안에 아래와 같은 행을 반드시 포함한다.

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

Linux는 또한 -t선택항목을 사용하여 1분간의 기정기다림시간을 재정의한다. shutdown은 아래와 같은 방법으로도 사용될수 있다.

shutdown 17:30	17시 30분에 완료한다
shutdown -r now	즉시 완료하고 재기동한다

사용자에게 체계끄기만을 허락하기

shutdown지령은 오직 상급사용자에 의해서만 실행될수 있다. 그러나 그는 때때로 자기가 없을 때 이 기능을 수행할수 있도록 대리사용자(backup user)에게 뿌리접근을 넘겨 주어야 할 필요가 있다. 그러나 이 사용자가 쉘에 《들어 갈수》 있게 해서는 절대로 안된다. 이것은 상급사용자가 뿌리와 같은 등록자리(account)를 또 하나 만들어야 한다는것을 의미한다. 많은 체계관리자들을 성가시게 하는 이 문제를 해결하는 방법을 여기에서 보자.

상급사용자의 능력은 /etc/passwd안에 있는 하나의 간단한 항목(entry)으로부터 나온다. 이것은 UID 0을 가지는 유일한 사용자이다. 방법은 먼저 일반사용자등록자리(말하자면 shut)를 useradd를 리용하여 만드는것이다.

```
seradd -u 210 -g users -s /bin/sh -d /home/shut -m shut
```

그다음 /etc/passwd에서 UID를 210으로부터 0으로 바꾸어 이 사용자에게 뿌리상태를 주어야 한다. useradd는 뿌리사용자ID의 재리용을 허락하지 않을수도 있으며 따라서 수동으로 /etc/passwd를 편집해야 할수도 있다. 이제는 shut의 .profile안에 shutdown지령을 배치(더 좋기는 exec와 함께)하여 shut가 그밖의 다른것은 할수 없게 한다. 이 기능은 명백하게 문서화되지는 않았지만 UNIX체계들에서 아주 잘 동작한다. 추가적으로 GUID를 0으로 설정하는것도 필요할수 있다.

이 방법의 결함은 뿌리등록자리에로 몰래 들어 오려고 시도하는 사람이 이 일을 몇초동안에 실현할수 있게 한다는데 있다. UID(또는 통과암호)를 바꾸는것을 제외하고는 이 등록자리의 권한을 조종할수 있는 방법은 없다.

22.7 플로피디스크의 조작

테프가 가장 보편적인 여벌복사(backup)장치이기는 하지만 탁상체계상에서는 플로피디스크들이 광범히 쓰이고 있다. 디스케트는 직장과 집의 기계들사이에서 파일들을 주고 받는데 가장 편리한 매체이다. 여기서는 실례로 3.5인치 1.44MB디스케트를 사용하고 있다.

22.7.1 플로피디스크의 초기화(format와 fdformat)

플로피를 여벌복사목적으로 사용하기전에 먼저 그것을 초기화해야 한다. 이것은 장치이름을 인수로 하여 format 또는 fdformat를 써서 수행할수 있다.

```
format /dev/rdsk/f0q18dt                    System V
```

이 지령은 1.44MB플로피를 초기화한다. System V는 초기화를 위하여 문자형장치를 사용한다. 초기화프로쎄스후에는 오유를 찾아 내는 검사가 뒤따른다.

Solaris에서 format지령은 구획을 만드는데 리용된다. Solaris에서 디스케트를 초기화하기 위하여서는 fdformat를 사용한다.

```
# fdformat
Press return to start formatting floppy
```

-d선택항목은 DOS초기화를 사용한다.

Linux

Linux도 플로피를 초기화하는데 fdformat지령을 사용한다. 장치이름은 다음과 같이 지정되여야 한다.

```
fdformat /dev/fd0H1440
```

22.7.2 플로피디스크의 복사(dd)

dd(disk dump)는 여러가지 과제를 수행할수 있는 다용도지령이다. 그것은 좀 구식이지만 그의 려과기능의 일부를 다른 UNIX도구들이 인계 받았다. 이 지령은 임의의 사용자가 불러 낼수 있으나 실제상 관리자의 도구이다. <선택항목=값>형식으로 된 선택항목계렬들을 가지고 있는 류다른 지령행이다.

dd는 파일체계를 복사하는데 널리 사용되였지만 오늘날에는 그의 역할이 주로 플로피나 테프와 갈

은 매체복사에만 쓰이고 있다. 대화식이 아니며 조작을 완성하기 위해서는 한조의 dd지령이 필요하다.

그러면 dd를 사용하여 1.44MB플로피디스크를 복사하여 보자. 첫 단계는 디스크상에 플로피의 영상(image)을 만드는것이다.

```
# dd if=/dev/rdsk/f0q18dt of=$$ bs=147456
10+0 records in
10+0 records out
```

예약어들은 if=(입력파일이름), of=(출력파일이름), bs=(블로크크기)이다. 우의 지령은 147456의 블로크크기(사실상 1.44MB디스케트용량의 10분의 1)를 사용하여 1.44MB플로피의 내용을 그대로 림시파일 $$에 복사한다.

다음은 원천플로피를 구동기에서 꺼내고 초기화된 목적플로피를 삽입한다. 첫번째 항목과 두번째 항목을 반전시킨 dd지령이 디스케트상에 이 림시파일을 복사한다.

```
# dd if=$$ of=/dev/rdsk/f0q18dt bs=147456
10+0 records in
10+0 records out
```

기동플로피들은 이러한 방법으로 복사해야 한다. 같은 방법으로 테프를 복사할수 있다. 두개의 테프구동기가 있다면 하나의 dd지령으로 그 일을 할수 있다.

```
dd if=/dev/rct0 of=/dev/rct1 bs=9k
```

주해

dd는 문자형장치 즉 /dev/rdsk안에 있는 장치나 /dev/rdiskette 또는 /dev/rct0과 같이 r로 시작하는 /dev안의 장치들에서만 사용한다. Linux는 두 방식을 위한 장치를 따로 가지지 않으며 자동적으로 뒤에 있는 방식을 선택한다.

22.7.3 DOS디스크의 조종

탁상우에서 Windows와 UNIX를 둘 다 보는것은 이제는 아주 례사로운 일로 되였다. 오늘날 UNIX는 DOS디스크에 읽기쓰기할수 있는 지령들을 제공한다(표 22-1). SVR4에서 이 지령들은 문자렬 dos로 시작하며 그뒤에는 류사한 기능을 수행하는 UNIX지령이 놓인다.

표 22-1. DOS지령들의 묶음(괄호안에 Linux지령이름을 주었다.)

지 령	동 작
doscp /dev/fd0135ds18:/tags	DOS디스크로부터 tags를 복사한다(mcopy)
doscat a:readme a:setup.txt	DOS디스크안에 있는 파일 readme 와 setup.txt를 련결시킨다(mtype)
dosdir /dev/dsk/f0q18dt	DOS디스크안의 파일들을 DOS형으로 렬거한다(mdir)
dosls /dev/dsk/f0q18dt	파일들을 UNIX의 ls형으로 렬거한다
dosmkdir a:bin	DOS디스크상에 등록부 bin을 생성한다(mmd)
dosrmdir a:bin	DOS디스크상에서 등록부 bin을 삭제한다(mrd)
dosrm /dev/dsk/f0q18dt:setup.inf	DOS디스크상에서 파일 setup.inf를 삭제한다(mdel)
dosformat b:	DOS체계상에서의 사용을 위하여 기동불가능한 구동기에서 디스크를 초기화한다(mformat)

가장 많이 쓰이는 지령은 doscp이며 이것은 디스크사이에 파일들을 복사한다.

```
doscp emp.lst /dev/dsk/f0q18dt:/per.lst
```

목적지정에는 두점(:)으로 구분된 2개의 요소 즉 장치이름(1.44플로피구동기)과 파일이름(/per.lst)이 있다. cp에서와 같이 다중파일복사도 가능하다.

```
doscp emp[123].lst /dev/dsk/f0q18dt
```

doscat는 지령행에서 인수의 간단한 《련결》을 수행한다. 2개이상의 파일이름이 지정되였을 때에는 매 파일을 위한 표준출력이 련결된다.

```
doscat /dev/dsk/f0q18dt:/CHAP01 /dev/dsk/f0q18dt:/CHAP02 > newchap
```

UNIX와 같이 DOS도 뿌리등록부를 가지는 계층파일체계구조로 되여 있다. DOS파일들도 UNIX에서의 행마감문자가 LF(line feed - 8진수로 012)이라는것을 제외하고는 UNIX파일과 류사하다. DOS에서 행마감문자는 CR(carriage return - 8진수 015)와 LF의 결합이다. 이 지령들에 의하여 파일들이 복사되고 현시될 때 적절한 변환이 진행된다. doscp와 doscat도 둘 다 행바꾸기가 없이 파일들을 복사하거나 련결하는 경우에는 -r선택항목과 함께 작업한다.

일부 기계들에서 이 DOS지령들은 그 기계에 DOS를 위한 구획이 따로 있다면 DOS용 하드디스크구획에서 작업할수도 있다. 고정디스크를 위한 장치이름은 체계에 의존하며 일부 체계들은 c:와 d:를 사용한다. DOS구획의 뿌리등록부안에 있는 파일들을 보려면 SCO UNIX체계상에서는 다음지령들중 아무것이나 리용할수 있다.

```
dosdir c:
dosdir /dev/hd0d                DOS구획의 xenix장치이름
dosdir /dev/dsk/0sC
```

표 22-1에 여러가지 장치이름을 가지고 이 지령들을 사용하는 실례를 보여 준다. a:와 b:가 동작하지 않는다면 그때는 /dev 혹은 /dev/dsk안에 있는 적당한 파일을 사용하시오.

Linux

Linux의 DOS형지령들은 문자 m으로 시작하며 나머지문자렬은 대응하는 DOS지령과 같다.

```
mcopy emp.lst a:
mcopy emp[1-3].lst a:

mdir a:
mdel a:*.txt
```

Linux가 DOS구동기이름을 사용한다는데 주의를 돌려야 한다. 이 지령들은 모두 《mtools》그룹에 속한다. 상세한 정보는 man mtools를 사용하여 얻을수 있다.

22.8 입출력복사(cpio)

규칙적인 여벌복사를 해두는것이 가지는 중요성은 보통 폭주가 일어 나고 다량의 자료가 손실될 때에 가서야 인정된다. 관리자는 체계안에 존재하는 자료의 안전성에 부분적으로 책임이 있다. 어느 파일들을 여벌복사시킬것인가를 정하고 그러한 여벌복사의 기간을 결정하는것은 그의 임무의 한부분이다. 여벌복사의 효과성은 손실 또는 손상된 자료파일들을 쉽게 보관시키는 관리자의 능력에 의하여 결정된다.

오늘날에는 cpio와 tar가 가장 보편적인 여벌복사프로그람이다. 이 두 프로그람은 파일들을 하나의 보존파일(archive)에 결합한다(매 파일의 내용앞에 머리부를 추가하여). 이 지령들은 일정한 우점이 있

으며 자료기록을 위해 사용하는 형식은 전혀 호환되지 않는다. tar보존파일은 때때로 cpio에 의해 읽혀질수 있지만 그 반대로는 되지 않는다.

여벌복사장치는 자기테프나 카세트테프, 플로피디스크 심지어 디스크파일일수도 있다. 작은 체계들은 테프설비를 가지고 있지 않을수도 있으며 따라서 여기서는 두 지령의 특징을 설명하는데 플로피구동기를 사용한다.

주해

Solaris기계상에서는 cpio와 tar지령을 써보기전에 /etc/init.d/volmgt stop을 써서 기록권관리데몬을 금지시켰는가를 확인해야 한다. CD-ROM을 자동적으로 태우는 이 데몬(vold)은 start인수를 가지고 그 지령을 사용하면 다시 능동상태로 될수 있다. 이 데몬을 정지하는 경우 그 구동기안에 디스크가 있어서는 안된다.

cpio지령(Copy Input-Output)은 파일들을 여벌복사장치에 복사하거나 여벌복사장치에서 파일을 복사해 온다. 파일이름목록을 얻기 위해 표준입력을 사용한다. 그다음 그것들을 내용 및 머리부와 함께 파일이나 장치에로 출력절환될수 있는 흐름에 복사한다. 이것은 cpio가 방향절환(redirection)과 관련결(piping)을 사용한다는것을 의미한다.

cpio는 2개의 주요선택항목 즉 -o(출력)와 -i(입력)를 사용하며 그중 하나는 지령행에 있어야 한다. 모든 다른 선택항목들은 이 주요선택항목들중의 어느 하나와 함께 사용되여야 한다. 이 절에서의 실례들은 System V장치이름들을 사용한다. Linux사용자들은 /dev/fd0H1440을 사용해야 하며 Solaris사용자들은 장치이름으로서 /dev/rdiskette를 사용해야 한다.

22.8.1 파일의 여벌복사(-o)

cpio는 표준입력만을 사용하므로 ls지령을 리용하여 cpio에로 입력시킬 파일이름목록을 발생할수 있다. -o주요선택항목은 표준출력에 보존파일을 만드는데 그러자면 장치파일에로 방향절환을 해야 한다. 1.44MB 플로피에 현재등록부의 파일들을 복사하는 방법을 아래에 준다.

```
# ls | cpio -ov > /dev/rdsk/f0q18dt          Solaris에서 /dev/rdiskette를 사용한다
array.pl
calendar
cent2fah.pl
convert.sh
xinitrc.sam
276 blocks               보존파일의 최대크기
```

-v선택항목은 매 파일의 복사가 끝날 때마다 그 파일이름이 현시되도록 한다. cpio가 입력해야 하는 것은 여벌복사되는 파일의 목록(한행에 한 파일씩)이다. 만일 이 목록이 파일안에 있다면 방향절환을 사용할수도 있다.

```
cpio -o >/dev/rdsk/f0q18dt < flist
```

증가여벌복사(incremental backup)

find는 파일목록을 만들수도 있다. 따라서 그의 선택기준을 만족시키는 파일들이 여벌복사될수도 있다. 선택된 파일들을 여벌복사하기 위해 find와 cpio를 결합하여 사용할 필요가 자주 있게 된다. 실례로 마지막 이틀동안에 변경된 파일들에 대하여 보기로 하자.

```
find . -type f -mtime -2 -print | cpio -ovB > /dev/rdsk/f0q18dt
```

find의 경로목록이 점(.)이므로 파일들은 상대경로이름으로 여벌복사된다. 그렇지만 그것이 /이면 절대경로이름이 사용된다.

-B선택항목은 입출력을 위한 블로크크기를 5120byte(기정크기의 10배)로 설정한다. 더 큰(또는 더 작은) 크기를 위해서는 -C선택항목을 사용해야 한다.

```
ls *.pl | cpio -ovC51200 > /dev/rdsk/f0q18dt             기정값의 100배
```

다중기록권여벌복사(multivolume backup)

여벌복사장치안에 만들어 진 보존파일이 그 장치의 용량보다 더 클 때 cpio는 구동기에 새 디스케트를 넣을것을 요구한다.

```
# find . -type f -print | cpio -ocB > /dev/rdsk/f0q18dt
Reached end of medium on output.
If you want to go on, type device/filename when ready
/dev/fd0
3672 blocks
```

cpio가 입력을 림시 정지하면 장치이름을 기입한다. 기계상에 두개의 플로피장치가 있다면 두 장치이름중 한개를 쓸수 있다. 그러면 디스케트가 바뀌여도 cpio는 복사를 계속한다. 이렇게 하여 보존파일은 몇개 크기(기록권)로 갈라 질수 있다.

22.8.2 파일의 재보관(-i)

완전한 보존파일 또는 선택된 파일들은 -i선택항목을 써서 재보관될수 있다. 파일들을 재보관하기 위해서는 입력절환을 사용하여 장치로부터 입력을 얻는다.

```
# cpio -iv < /dev/rdsk/f0q18dt
array.pl
calendar
cent2fah.pl
convert.sh
xinitrc.sam
276 blocks
```

보조등록부들을 재보관할 때 cpio는 그 보조등록부구조가 하드디스크안에 이미 있다고 가정한다. 보조등록부들이 존재하지 않는 경우 그것들을 만들수 없다. 그러나 -d(등록부)선택항목으로 그것을 재정의할수 있다. cpio에서는 인용부호안에 통용기호(wild-card)를 쓸수 있으므로 이 기호를 써서 여러개의 파일들을 재보관할수 있다.

```
cpio -i "*.sh" < /dev/rdsk/f0q18dt
```

참고

파일은 그 경로이름에 해당한 등록부에 재보관된다. 즉 파일을 여벌복사할 때 절대경로이름(례/home/romeo/unit13)이 사용되였다면 그 파일은 한 등록부(/home/romeo)에만 재보관된다. 그러나 상대경로이름이 리용되였을 때는 임의의 위치에 보관될수 있다. 관리자가 종종 한 등록부로부터 파일들을 여벌복사하여 다른 곳에 그것들을 재보관하려고 할 때에는 보통 상대파일이름을 리용하는것이 더 좋을것이다. 경로이름지정에 /이 아니라 점을 가진 find를 사용하고 있는가를 확인하여야 한다.

변경시간조절(-m)

기정으로 파일이 보존파일로부터 풀릴 때에는 그 변경시간이 풀린 시간으로 설정된다. 이것은 파일이 재보관된후에 변경되지 않았음에도 불구하고 앞으로의 증가여벌복사에 참여할수 있는 문제를 일으킨다. 그러한 결과에 대해서는 touch를 써서 변경시간을 바꾸어야 할것이다. 다른 방도로서 -m선택항목을 써서 cpio가 변경시간을 유지하도록 할수 있다.

cpio는 매체의 파일변경시간과 디스크의 파일변경시간을 비교한다. 만일 디스크파일이 복사본보다 더 새롭다면 보관하지 않으며 다음과 같은 통보문을 내보낸다.

```
"current <unit14> newer"
```

이것은 파일의 최종판본을 유지하는 쓸모 있는 내부보호기능이다(tar는 이 기능을 가지지 않는다). 그러나 -u선택항목을 써서 재정의할수 있다.

참고

한 기계에서 다른 곳으로 파일들을 옮길 때 대체로 tar대신 cpio를 사용하는것이 더 좋다. 한 기계에 있는 새로운 파일은 이전 파일로 덧쓰기되지 않게 한다.

22.8.3 보존파일의 현시(-it)

-t선택항목은 파일들을 재보관하지 않고 장치내용을 현시한다. 이 선택항목은 -i선택항목과 결합되여야 한다.

```
# cpio -itv < /dev/rdsk/f0q18dt
100755 henry    605  Oct 28 23:34:07 1997  cent2fah.pl
100755 henry    273  Oct 18 23:34:07 1997  check_number.pl
100755 henry    531  Oct 18 23:34:08 1997  dec2bin.pl
100755 henry    214  Oct 18 23:34:08 1997  get_home.pl
```

파일들이 목록의 형식으로 현시된다(Linux와 Solaris의 출력이 동일하다). 이 형식은 그 파일의 변경시간처럼 허가권의 8진표현을 보여 준다.

22.8.4 기타 선택항목들

-o와 -i방식들에서 사용될수 있는 3개의 중요한 선택항목이 있다.

- -r선택항목은 복사프로쎄스를 시작하기전에 매 파일의 이름을 바꾸게 한다. 체계는 매 파일이름을 보여 주고 응답을 요구한다. 만일 파일이름을 입력하면 그 파일로 복사가 수행되며 응답이 없으면 그 파일은 복사되지 않는다.
- -f선택항목은 cpio가 식에 있는것들을 제외한 모든 파일을 선택하게 한다.

```
cpio  -ivf  "*.c"  </dev/rdsk/foq18dt
```

- -c선택항목은 cpio가 머리부를 만들기 위해 2진형식이 아니라 ASCII문자들을 사용하게 한다. 여러가지 기계들에서 매체를 사용할 때 편리한 여벌복사를 만들려면 이 선택항목을 선택해야 한다.

cpio선택항목을 표 22-2에서 보여 준다. cpio는 파일목록을 제공해 주는 또 다른 지령(보통 find)이나 파일에 의존한다. 그것은 지령행에서 파일이름인수들을 받아 들일수는 없다. cpio보존파일은 그 지령

을 호출할 때마다 재정의된다. 파일을 보존파일에 추가시키는 방법은 없다(Solaris와 Linux는 제외). 이것은 tar가 수행한다.

표 22-2. cpio의 일반선택항목들(-i 또는 -o와 함께 사용)

선택항목	의 미
-d	필요에 따라 등록부를 생성한다
-c	이식성을 위하여 머리부정보를 ASCII문자형식으로 써넣는다
-r	대화적방법으로 파일들의 이름을 고친다
-t	보존파일안의 파일들을 렬거한다(-i선택항목과만 리용)
-u	새로운 파일을 이전 파일로 덧쓰기한다
-v	불필요한 항목—복사되고 있는 파일들의 목록을 현시한다
-m	원래파일의 변경시간을 유지한다
-f exp	exp안의 파일들을 제외한 모든 파일들을 복사한다
-Csize	입출력블로크의 크기를 size바이트로 설정한다
-A -O device	device에 파일들을 추가한다(Solaris와 Linux에서만 유효)
-H tar	tar머리부형식을 읽거나 생성한다(Solaris와 Linux에서만 유효)
-E file	file에 렬거된 파일들만을 풀어 낸다(Solaris와 Linux에서만 유효)

22.9 테프보존프로그람(tar)

tar(tape archive)지령은 cpio가 출현하기전부터 존재하여 왔다. 오늘날 그것은 테프상에도 보존파일을 만들며 플로피에서도 만든다. cpio와 달리 tar는 보통 표준출력에 쓰지 않고 매체안에 보존파일을 만든다. tar는 cpio에서 찾아 볼수 없는 몇가지 훌륭한 기능을 가진 다용도지령이다.

- 파일목록을 얻기 위해 표준입력을 사용하지 않는다. tar는 파일이름과 등록부이름을 인수로 받아들인다.
- 하나이상의 완전한 등록부나무를 복사한다. 즉 이 지령은 기정에 의해 재귀적으로 동작한다.
- 한개의 보존파일안에 같은 파일을 여러개 만들수 있다(Solaris와 Linux의 cpio도 가능).
- 전체 보존파일에 덧쓰기함이 없이 보존파일에 추가할수 있다(Solairs와 Linux의 cpio도 가능).

tar는 몇개의 주요선택항목들중 어느 하나와 함께 사용되며 일반적인것들은 -c(copy), -x(extract), -t(list)이다. 장치이름을 지정하려면 추가적으로 -f선택항목이 사용되여야 한다. tar선택항목을 표 22-3에서 보여 준다.

22.9.1 파일의 여벌복사(-c)

tar는 지령행에서 직접 등록부와 파일이름을 받아 들인다. 파일들을 여벌복사장소에로 복사하기 위하여 -c주요선택항목을 사용한다.

```
# tar -cvf /dev/rdsk/f0q18dt /home/sales/SQL/*.sql
a /home/sales/SQL/invoice_do_all.sql 1 tape blocks
a /home/sales/SQL/load2invoice_do_all.sql 1 tape blocks
a /home/sales/SQL/remove_duplicate.sql 1 tape blocks
a /home/sales/SQL/t_mr_alloc.sql 10 tape blocks
```

표 22-3. tar선택항목

주요선택항목(오직 한개만이 사용된다) **선택항목**	 **의 미**
-c	새로운 보존파일을 만든다
-x	보존파일로부터 파일들을 풀어 낸다
-t	보존파일의 내용을 렬거한다
-r	보존파일의 끝에 파일들을 추가한다
-u	r와 류사하나 파일들이 보존파일안에 있는것들보다 더 새로운 파일일때에만 추가한다

일반선택항목 **선택항목**	 **의 미**
-f device	기정값대신에 장치이름으로서 경로이름 device를 사용한다
-v	불필요한 항목－완전한 형식으로 파일들을 렬거한다
-w	주어 진 동작에 대하여 사용자에게 확인한다
-b n	블로크화인자 n을 사용한다(n은 20까지로 제한된다)
-m	파일의 변경시간을 풀어 낼 때의 시간으로 바꾼다
-I file	파일이름들을 file로부터 얻는다(Solaris에서만)
-T file	파일이름들을 file로부터 얻는다(Linux에서만)
-k num	다중기록권여벌복사－volume의 크기를 num키로바이트로 설정한다(Solaris에서만)
-M	다중기록권여벌복사(Linux에서만)
-z	gzip를 리용한 압축 및 해제(Linux에서만)
-Z	compress를 리용한 압축 및 해제(Linux에서만)
-X file	file안에 있는 파일이름들을 제외한다(Solaris와 Linux에서만)

이것은 모든 SQL스크립트들을 절대경로이름과 함께 플로피디스크에 여벌복사한다. 매 경로이름의 앞에 있는 단일문자 a는 파일이 추가된다는것을 가리킨다. -v선택항목은 매 파일에서 사용된 블로크의 수를 보여 준다.

주해 tar는 선택항목들의 조작에서 아주 자유롭다. tar cvf는 tar -cvf와 같으며 -기호를 전혀 요구하지 않는다. 그러나 tar의 앞으로의 판본은 이것을 지원하지 않을것이다.

절대경로이름을 쓰는 방법으로 파일들을 복사할 때는 같은 제한이 적용된다. 즉 파일들이 같은 등록부에만 보관된다. 그렇지만 다른 등록부에 파일들을 설치하려면 먼저 /home/sales/SQL에로 등록부를 바꾸고 그다음 상대경로이름을 사용하여야 한다.

```
cd /home/sales/SQL
tar -cvf /dev/rdsk/f0q18dt ./*.sql              /을 사용한다
```

블로크크기를 18로 하여 리용한다면 지령이 더 빨리 실행될것이다(즉 18×512byte).

```
tar -cvfb /dev/rdsk/f0q18dt 18 *.sql              /는 실지 쓰지 않는다
```

-f와 -b는 둘 다 인수가 뒤따르므로 선택항목문자렬 -cvfb뒤에 있는 첫 단어(/dev/rdsk/foq18dt)는 -f를 위한 인수이고 둘째 단어(18)는 -b에 대응된다.

tar와 cpio를 기정블로크크기로 사용하지 않는것이 좋다. 체계가 허용하는만큼 값을 높이 선택하시오. 블로크크기를 더 크게 하면 입출력조작의 속도가 높아 진다.

tar의 우점은 모든 보조등록부들을 포함한 전체 등록부나무를 복사할수 있다는데 있다. 현재의 등록부가 숨은 파일과 함께 또는 숨은 파일이 없이 여벌복사될수 있다.

```
tar -cvfb /dev/rdsk/f0q18dt 18 *        숨은 파일을 여벌복사하지 않는다
tar -cvfb /dev/fd0 18 .        숨은 파일도 여벌복사한다
```

여기서 파일들은 상대경로이름으로 여벌복사된다. 모든 파일이 하나의 디스케트에 들어 갈수 없다면 System V의 tar는 그것들을 가능한껏 많이 복사하고 그다음 경고없이 프로그람을 탈퇴할수도 있다.

깊이가 있는 재귀적인 등록부구조를 복사할 때 한가지 문제점이 있다. tar는 UNIX파일이름이 14문자로 제한되였을 때 개발되였다. 오늘날에도 tar는 100문자를 넘는 경로이름은 조작할수 없다. 등록부구조에서 큰 깊이를 가지는 현대의 UNIX체계상에서는 이것이 일련의 제한점을 준다는것이 알려 졌다. Solaris는 더 큰 한계를 가지며 -E선택항목을 써서 그것을 더 확장한다.

다중기록권여벌복사(-k)

다중기록권디스케트여벌복사를 위하여 Solaris의 tar는 특수한 선택항목(-k)을 사용하며 그뒤에 키로바이트단위로 기록권크기를 준다. 아래에 SCO UNIX에서 파일 index를 여벌복사하는 방법을 주었다.

```
# tar -cvfkb /dev/rdsk/f0q18dt 1440 18 index
Volume ends at 1439K, blocking factor = 18
tar: large file index needs 2 extents.
tar: current device seek position = OK
+++ a index 1439K [extent #1 of 2]
```

tar에서는 2개의 1440KB(-k의 인수)디스케트가 요구된다. 첫 기록권이 채워 진후 tar는 새로운 디스케트를 요구한다.

```
tar: please insert new volume, then press RETURN.
```

보관시에도 같은 선택항목이 사용되여야 한다.

22.9.2 파일의 재보관(-x)

주요선택항목 -x(extract)를 가지고 파일들을 재보관한다. 파일이나 등록부이름이 지정되지 않았을 때는 여벌복사장치로부터 모든 파일을 재보관한다. 다음의 지령이 여벌복사된 파일들을 보관한다.

```
# tar -xvfb /dev/rdsk/f0q18dt 18
x /home/sales/SQL/invoice_do_all.sql, 169 bytes, 1 tape blocks
x /home/sales/SQL/load2invoice_do_all.sql, 456 bytes, 1 tape blocks
x /home/sales/SQL/remove_duplicate.sql, 237 bytes, 1 tape blocks
x /home/sales/SQL/t_mr_alloc.sql, 4855 bytes, 10 tape blocks
```

하나이상의 등록부나 파일이름을 제공하여 선택적인 추출을 할수 있다.

```
tar -xvf /dec/rdsk/f0q18dt tulec1 project2
```

cpio와 달리 파일들이 추출될 때 파일들의 변경시간도 유지된다. 이것은 추출할 때 체계시간을 반영하도록 -m선택항목을 써서 재정의할수 있다.

주해

cpio와 달리 tar의 일부 판본(Solaris에서와 같은)들은 통용기호를 해석하지 않는다. tar -xvf /dev/fd0 *.pl을 사용하면 쉘은 그 기호로 이루어 진 파일을 찾는다. 다시말하여 현재등록부에 그 파일이 존재하여야 한다는것이다. 많은 경우에 파일들이 존재하지 않으며 따라서 추출은 완성되지 않을수 있다. 그러므로 지령행에 파일이름들을 명확히 지정해야 한다.

주의

다중기록권여벌복사로 재보관할 때 정확한 기록권으로부터 시작하는가를 확인해야 한다. tar는 정확한 기록권으로부터 시작하라고 지적하지도 않으며 정확한 기록권으로 차례차례 삽입되는가도 알려 주지 않는다. 주의하지 않으면 파일들이 쉽게 류실될수 있다.

22.9.3 보존파일의 현시(-t)

cpio와 같이 주요선택항목 -t는 파일들을 보관함이 없이 장치의 내용을 간단히 현시한다. -v선택항목과 결합할 때는 아래에 렬거된것과 같은 긴 형식으로 현시한다.

```
# tar -tvf /dec/rdsk/f0q18dt
rwxr-xr-x203/50    472 Jun  4 09:35 1991 ./dentry1.sh
rwxr-xr-x203/50    554 Jun  4 09:52 1991 ./dentry2.sh
rwxr-xr-x203/50   2299 Jun  4 13:59 1991 ./func.sh
```

여기에 주의를 돌릴 필요가 있다. 이 파일들은 상대경로이름을 가지고 여벌복사되였다. 매 파일이름의 앞에는 ./가 붙어 있다. 이것을 생각하지 않고 그 디스케트로부터 파일 func.sh을 풀려면 다음과 같이 할수 있다.

```
# tar -xvf /dev/fd0 func.sh              일반플로피장치를 사용한다
tar: func.sh: Not found in archive
```

파일이 func.sh가 아니라 ./func.sh로 존재하기때문에 tar는 그 파일을 찾지 못한다. 파일이름앞에 ./를 붙여 다시 해보자. 우와 같은 추출오유를 만날 때에는 언제나 이것을 생각하시오.

22.9.4 tar를 압축도구와 함께 사용하기

tar가 항상 장치이름과 함께 사용되는것은 아니다. 즉 -f선택항목을 쓰면 일반파일이름과 함께 사용될수도 있다. 이것은 파일들을 하나의 디스크파일로 묶을수 있다는것을 의미한다. tar가 재귀적으로 동작하므로 전체 등록부나무를 보관하는데 사용할수 있다.

```
tar -cvf quotes.tar quotes.dir           quotes.dir는 등록부이다
tar -cvf quotes.tar *
```

첫째 지령은 인수로서 등록부이름을 사용하며 둘째 지령은 현재등록부의 모든 파일들을 사용한다. 두 경우에 여러개의 파일들을 포함하는 quotes.tar파일이 생성된다. 그러면 이 파일을 다른 사용자에게 보낼수 있고 그의 telnet나 ftp를 자기의 등록자리에 놓지 않고도 자기의 파일들과 등록부들을 가지고 그가 작업할수 있게 된다. 보통 파일들은 전송되기전에 compress나 gzip로 압축한다. 이 지령의 출력은 확장자 .Z 또는 .gz를 가지는 같은 파일이름으로 다시 씌여 진다.

```
compress quotes.tar          quotes.tar.z를 만든다
gzip quotes.tar              quotes.tar.qz를 만든다
```

이것은 생소하게 보일수도 있지만 cpio 와 tar, 압축도구들인 compress와 gzip는 려파기로서 동작할수도 있다. 묶고 압축하는 동작을 관흐름에서 결합할수 있다.

```
tar -cvf - quotes.dir | compress > quotes.tar.Z
tar -cvf - quotes.dir | gzip > quotes.tar.gz
```

여기서 .tar, .Z, .gz확장자를 고의적으로 제공하였다는데 주의할 필요가 있다. compress와 gzip는 둘 다 다중압축형식을 취급할수 있으므로 이 확장자들을 찾는다. 이 파일은 elm, pine, Netscape와 같은 우편프로그람의 첨부물로서 배포될수 있다.

수신종점에서 그 첨부물은 같은 확장자를 가진 파일로 보존되여야 한다. 추출프로쎄스는 류사하지만 반대방법으로 작업한다. 아래에 uncompress를 리용하여 작업하는 례를 보여 준다.

```
uncompress quotes.tar.Z      quotes.tar를 만든다
tar -xvf quotes.tar          등록부 quotes.dir를 다시 얻는다
```

tar-gzip로 압축된 파일도 류사한 방법으로 풀수 있다.

```
gunzip quotes.tar.gz         quotes.tar를 만든다
tar -xvf quotes.tar          등록부 quotes.dir를 다시 얻는다
```

보존파일을 만들 때 중간파일을 만들지 않고 관흐름을 사용하였다. 그와 류사하게 추출에서도 관흐름을 사용할수 있다.

```
unompress -c quotes.tar.Z | tar -xvf -
gunzip -c quotes.tar.gz | tar -xvf -
```

본문을 포함하는 큰 파일들은 더많이 압축된다. GIF와 JPEG파일들은 이미 압축형식으로 자료를 가지고 있기때문에(6.6) 압축효과가 적게 나타난다. tar는 표준입력과 표준출력을 둘다 표현하기 위해 -를 사용하며 흐름을 지정하기 위해 주요선택항목(-c나 -x)을 추가적으로 사용한다.

22.9.5 기타 선택항목

tar에는 몇개의 선택항목들이 더 있다.

- -r주요선택항목은 보존파일에 파일을 추가하기 위해 사용된다. 차이점은 하나의 보존파일이 여러 판본의 같은 파일을 포함할수 있다는것이다.
- -u주요선택항목도 보존파일에 파일을 추가할수 있으나 단지 파일이 이미 존재하지 않거나 새로운 판본으로 교체될 때뿐이다.
- -w선택항목은 대화식복사와 재보관을 할수 있다. 파일이름을 현시하고 주어 진 동작을 요구한다(y 혹은 n).
- tar의 일부 판본들은 파일로부터 파일이름들을 얻어 내는 특별한 선택항목을 사용한다. 100개이상의 파일의 목록을 가지고 있어서 지령행에 입력하는것이 어렵거나 불가능할 때 이 기능을 사용할수 있다. 이 선택항목은 표준이 아니며 Solaris는 -I를 사용하고 Linux는 -T를 사용한다.

참고

보존파일에 파일들을 추가하기 위해서는 주요선택항목-r를 사용하며 새로운 파일들만을 보관하려면 -u선택항목을 사용한다. tar는 하나의 보존파일에 한 파일을 여러개 복사하여 보관할수도 있다.

주의

cpio와 tar를 사용할 때 문자형장치가 사용되는가? 즉 /dev/rdsk안의것(/dev/dsk가 아닌)이거나 /dev안의 r로 시작되는 파일들인가를 확인해야 한다. 만일 블로크장치가 사용된다면 다중기록권여벌복사에서 문제가 생길수 있다. Linux사용자들은 걱정할 필요가 없다.

Linux

GNU tar지령은 System V의것보다 더 강력하며 특수한 선택항목들을 지원한다. System V에서 사용하는 선택항목들은 유감스럽게도 때때로 오유가 있다. 즉 호환성문제들이 종종 제기된다.

압축(-z와 -Z)

GNU tar는 파일들을 여벌복사하면서 압축기능을 지원한다. gzip로 압축할 때는 -z선택항목이, compress로 압축할 때는 -Z선택항목이 사용된다.

```
tar -cvzf /dev/rft0 .
tar -cvZf /dev/rft0 .
```

추출할 때나 -t로 현시할 때 -z나 -Z선택항목은 한가지로 사용된다.

다중기록권여벌복사(-M)

파일들이 한장의 플로피에 넣어 질수 없다면 tar는 경고를 내보낸다. 그러나 -M선택항목을 사용하면 더 지능적으로 동작한다.

```
# tar -cvf /dev/fd0H1440 -M *
.....
Prepare volume #2 for /dev/fd0H1440 and hit return:
```

tar는 장치이름에서 기록권규격을 식별한다. System V판의 tar에서는 이렇게 할수 없다.

22.10 디스크공간관리

디스크공간의 관리는 관리자의 중요한 기능의 하나이다. 하루에도 많은 파일들이 축적될수 있다(특히 등록부/tmp와 /var/tmp안에).이러한 축적이 검사되지 않으면 결국은 전체 디스크공간을 차지할수 있으며 체계기능의 속도저하를 초래할수 있다. 디스크공간을 감시하기 위하여 관리자는 df, du, find지령들을 사용한다. 그러면 관리자의 시점에서 이 지령들중 2개를 다시 고찰해 보자.

22.10.1 사용자에 의하여 소비된 공간을 알아내기(du -s)

체계안에서 동적공간의 대부분은 사용자의 홈등록부와 자료파일들에 의하여 소비된다. 매 사용자의 홈등록부에 대하여 통보하기 위해서는 /home/*인수를 가진 du -s를 사용해야 한다. 출력은 간단하지만 아주 유익하다.

```
# du -s /home/*
1204    /home/ftp          이름이 없는 ftp의 뿌리
144208  /home/henry
1536    /home/httpd        Web봉사자의 뿌리
98290   /home/image
```

```
28346   /home/sales
```

du는 또한 등록부안의 매 파일에 대하여 통보할수 있지만(-a선택항목) 그 목록은 너무 커서 다른데 사용할수 없다. 그대신 일부 엄중한 디스크공간랑비자들을 찾아 낼수 있으며 제한적인 통보가 필요한것이다. 이 일은 find가 더 잘 수행한다.

22.10.2 관리자의 도구(find의 재고찰)

find지령은 실천적으로 모든 속성화파일을 비교할수 있다. 여기서는 강력한 디스크관리도구로 되는 일부 선택항목들을 보겠다.

큰 파일의 위치를 알아 내기(-size)

find는 큰 파일들의 위치를 알아 내기 위해 -size연산자를 사용한다. 다중선택기준도 지정될수 있다.

```
find /home -size +2048 -print                1MB이상 파일들
find /home -size +2048 -size -8192 -print            1MB~4MB사이의 파일들
```

여기서 find는 512byte의 블로크크기를 사용한다. 또한 정확한 용량을 알고 있다면 파일을 추적하게도 한다. 만일 어디엔가 38765byte의 파일을 배치하였으나 그 위치를 기억하지 못할 때 아래와 같이 find를 리용할수 있다.

```
find / -size 38765c -print
```

사용되지 않는 파일들을 찾기(-mtime과 atime)

많은 파일들이 몇달 지어 몇년동안 호출되거나 변경되지 않는다. find의 -mtime과 -atime연산자는 파일의 변경시간과 접근시간을 쉽게 비교하여 그것들을 선택할수 있다. 아래에 관리자가 /home등록부를 규칙적으로 주사하여 1년동안 접근되지 않았거나 6개월동안 변경되지 않은 파일들을 찾아 낸다.

```
find /home -atime +365 -o -mtime +180 -print | mail root
```

증가여벌복사(-newer)

증가여벌복사를 하는데도 find를 사용할수 있다. 먼저 현재의 체계날자와 시간을 가진 0byte파일을 만들어야 한다. 다음의 여벌복사파일들은 이 파일보다 새로운 파일들을 선택한다. 아래의 스크립트행들은 그의 간단한 실현이다.

```
find /home -newer .last_time -print | cpio -o > /dev/rct0
touch .last_time
```

이 두개의 행은 임의의 시간에 함께 사용될수 있으며 touch는 마지막여벌복사시간이 파일의 변경시간자리에 보관되도록 한다. tar를 사용할수도 있다.

```
tar -cvf /dec/rct0 `find /home -type f -newer .last_time -print`
touch .last_time
```

type는 f로 지정되여야 한다. 그렇지 않으면 find는 등록부를 출력의 부분으로서 보여 주며 tar는 그것을 그 등록부의 모든 파일을 포함하는것으로 해석한다. 이것은 파일들이 2중으로 여벌복사될수 있다는것을 의미한다.

태워 진 파일체계들을 피하기(-mount)

뿌리파일체계를 여벌복사하기 위해서는 보통 다른 파일체계들이 내리워 질것을 요구한다. 지금은

find가 이 나무탐색을 제한하는 -mount예약어를 제공하므로 더이상 그렇게 할 필요가 없다. 하루이내에 변경된 뿌리파일체계의 파일들을 여벌복사하는것은 아주 쉽다.

```
find / -depth -mount -mtime -1 -print | cpio -ocvB -o /dev/rct0
```

이번에는 find가 다른 파일체계를 탐색하는 경우가 아니다. -depth예약어는 등록부안의 파일들이 등록부자체보다 먼저 동작되도록 한다. -mtime예약어가 생략되면 관호름렬이 뿌리파일체계의 완전한 여벌복사를 만든다. find선택항목을 표 7-5에서 보여 준다.

참고

체계설치후에는 즉시 테프에 뿌리파일체계를 여벌복사하여야 한다. 계속하여 앞으로도 뿌리체계파일들이 변경될 때마다 여벌복사를 해야 한다.

22.10.3 동적지령행의 구성(xargs)

find의 -exec연산자를 UNIX지령과 함께 사용할 때 한가지 문제가 있다. find가 삭제할 파일 200개의 목록을 만든다면 rm은 200번 실행되여야 한다. xargs는 rm으로 하여금 200개 파일이름을 인수로 하여 한번만 사용되게 함으로써 이 문제를 해결하여 준다.

xargs는 UNIX에서 잘 인정되지 않거나 리해되지 않는 지령들중의 하나이다. 그것은 표준입력으로 제공되는 자료를 목록으로 만들고 이 목록을 사용되는 지령에 그의 인수로서 제공한다. 다음의 지령들은 꼭 같은 동작을 수행하지만 두번째것이 훨씬 더 빠르다.

```
find /usr/preserve -mtime +30 -exec rm -f {} \;
find /usr/preserve -mtime +30 -print | xargs rm -f
```

xargs는 여기서 find로부터 파일목록을 얻고 rm에 인수들을 제공한다. 그러므로 find가 30개 파일을 선택하였다고 해도 rm은 오직 한번만 실행될것이다.

지령들은 보통 조작할수 있는 인수의 수를 제한한다. xargs는 -n선택항목을 사용하여 지령을 한번 호출할 때마다 지정된 개수의 인수를 제공한다.

```
find / -name core -size +1024 -print | xargs -n20 rm -f
```

만일 find가 100개 파일을 찾는다면 rm은 인수로써 한번에 20개 파일씩 5번 호출된다.

22.11 passwd를 리용한 통과암호관리

passwd지령은 통과암호관리를 돕기 위한 여러가지 선택항목을 제공한다. 사용자가 자기의 통과암호를 잊어 버렸거나 새로운 사용자에게 통과암호를 할당할 때 체계관리자는 그들이 지장없이 가입할수 있는 준비를 하고 그들에게 새 통과암호를 설정해야 한다. 이것은 통과암호를 완전히 지우는것으로써 간단히 수행된다.

```
passwd -d henry                  암호를 요구하지 않는다
```

이것은 위험하며 추가적인 문제를 일으킬수 있다. 즉 사용자는 어떤 때에는 전혀 통과암호를 설정할수 없을수도 있다. 따라서 관리자는 한번 자기가 설정하고 첫 가입시에 사용자가 그것을 바꾸게 할수도 있다. 이것은 -f선택항목을 리용하여 수행된다(Linux에서는 불가능).

```
passwd -f henry                암호는 가입시에 변화되여야 한다
```

UNIX체계들은 원래 통과암호들을 부호화된 형태로 /ect/passwd안에 보관하였다. 그 파일은 재간있고 장난기 있는 해커들에 의하여 해독되거나 공격 받을수 있는 약점이 있기때문에 현대체계들은 그 위치를 /etc/shadow로 옮겨 놓았다. shadow는 9개의 마당(부호를 포함)을 가지며 객관은 해독할수 없다. /etc/shadow안의 대부분의 마당은 적절한 선택항목을 가진 passwd를 사용하여 갱신할수 있다.

통과암호의 변경을 관리하기 위하여 shadow는 **통과암호로화**(password aging)라고 하는 정교한 체계도 제공한다. 예정된 날자수가 지나면 사용자가 자기의 통과암호를 변경하도록 할수 있다. 또한 사용자가 너무 자주 변경시키는것을 막을수도 있다.

```
passwd -n 14 julie
passwd -x 30 julie
```

-n 14는 julie가 2주일내에 자기의 통과암호를 바꾸는것을 막으며 한편 -x 30은 30일후에 바꾸도록 한다.

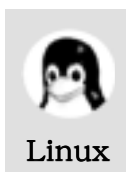
보안에서 흥미 있는것은 때때로 사용자등록자리를 잠그는것이 필요할수도 있다는것이다. passwd의 -l(lock)선택항목을 써서 이것을 실현할수 있다.

22.12 init에 의하여 사용되는 rc스크립트

init와 /etc/inittab는 체계가 기동하고 닫기는 방식을 완전히 조종한다. 더우기 체계가 실행준위를 바꿀 때 init는 새로운 실행준위에서 실행되여야 할 프로쎄스들과 실행되지 말아야 할 프로쎄스들을 찾기 위하여 inittab를 본다. 먼저 실행되지 말아야 할 프로쎄스들을 제거하고 그다음 실행되여야 할 프로쎄스들을 적재한다.

모든 inittab는 /etc 또는 /sbin안에 배치되여 있는 일부 **rc스크립트**(run command script)들의 실행을 지정한다. 이 스크립트들은 매 실행준위마다 각각 하나씩 대응되도록 rc0, rc1, rc2와 같은 이름을 가진다. 이것을 /etc/inittab안의 다음의 행들을 보면 명백히 알수 있다.

```
so:o:wait:/sbin/rc0                >/dev/console 2<>/dev/console </dev/console
s1:1:wait:/sbin/shutdown -y -iS -g0  >/dev/console 2<>/dev/console </dev/console
s2:23:wait:/sbin/rc2               >/dev/console 2<>/dev/console </dev/console
s3:3:wait:/sbin/rc3                >/dev/console 2<>/dev/console </dev/console
s6:6:wait:/sbin/rc6                >/dev/console 2<>/dev/console </dev/console
```

init는 실행준위에 해당한 스크립트를 실행시킨다. 그러나 rc2는 상태 2와 3에서 다 실행된다. 매 rc스크립트는 더 나아가서 등록부 /ect/rcn.d안에 있는 일련의 스크립트들의 실행을 지정한다. 이것은 실행준위 2에서는 init가 /etc/rc2를 실행시키고 그것이 다시 /etc/rc2.d안에 있는 스크립트들을 실행시킨다는것을 의미한다.

Linux

rc파일들

Linux에서의 초기화파일들은 원래 BSD에 기초하였지만 지금은 System V의 특징을 강하게 가진다. 그렇지만 rc파일들과 등록부들을 모두 /etc/rc.d안에 존재한다. 게다가 rcn을 사용하는 대신에 Linux는 아래에 보여 준것과 같은 여러가지 인수를 가지고 하나의 rc를 사

용한다.

```
10:0:wait:/etc/rc.d/rc 0
11:1:wait:/etc/rc.d/rc 1
12:2:wait:/etc/rc.d/rc 2
```

rcn.d등록부들속에 있는 모든 스크립트들은 /etc/rc.d/rc로부터 실행된다. Linux에서의 순서는 다음과 같다. 즉 실행준위 n으로 절환하기 위하여 init가 /etc/rc.d/rc n을 실행시키며 그것이 /etc/rc.d/rcn.d안에 있는 스크립트들을 실행한다.

시동스크립트와 제거스크립트

이제는 /etc/rcn.d등록부안의 스크립트들에로 주의를 돌려 보자. 이 등록부들은 /etc/rc3.d의 목록에 보여준바와 같이 두가지 형의 파일들을 보존한다.

```
k60nfs.server      S69inet              S75cron        S91agaconfig
k76snmpdx          S70uucp              S76nscd        S91leoconfig
k77dmi             S71rpc               S80lp          S92rtvc-config
S20sysetup         S73cachefs.daemon    S85power       S99audit
S21perf            S73nfs.client        S88sendmail    S99dtlogin
S30sysid.net       S74autofs            S88utmpd
```

이 rcn.d등록부안에 있는 스크립트들은 체계들을 구성하고 망을 설치하고 데몬들을 활성화하는것으로 체계를 완전히 초기화한다. 그것들은 2개의 묶음으로 실행된다. 체계가 실행준위 3으로 들어 갈 때 rc는 K로 시작되는 모든 스크립트(kill script)들을 stop인수와 함께 실행시킨다. 이것은 이 준위에서 실행되지 말아야 할 모든 프로쎄스들을 제거한다. 그다음 S로 시작되는 스크립트(start script)들을 start인수와 함께 실행한다. rc3스크립트를 보면 그것을 수행하는 2개의 for순환을 발견하게 될것이다.

스크립트는 둘이상의 실행준위에서 실행될수도 있으므로 스크립트들은 /etc/init.d안에 위치한 파일들을 가리키는 련결도 가지고 있다. 이 등록부안의 파일들은 3문자의 접두어가 붙은 류사한 이름들을 가지고 있다. 이것은 오직 한 장소에서만 파일을 수정하면 되는 깨끗한 배렬이다.

체계관리자는 이 스크립트들이 어떻게 작업하는가를 알아야 한다. 또한 해당한 봉사를 시작하는 스크립트를 식별할수 있어야 한다. 종종 파일이름들이 실마리를 제공하지만 때때로 스크립트의 위치를 알아 내기 위하여 grep가 필요하다. 아래에 인터네트데몬을 시동하는 스크립트를 식별하는 방법을 주었다.

```
# grep -l inet *
S69inet
S72inetsvc
```

주해

rcn.d안의 S로 시작되는 스크립트는 봉사를 시작한다는 의미이며 한편 K로 시작되는것은 봉사를 제거하기 위한것임을 의미한다. 제거스크립트는 시작스크립트보다 먼저 실행된다. 그것들은 모두 대응하는 /etc/rcn(Linux에서는 인수 n을 가지고 /etc/rc를)스크립트로부터 실행된다(n은 실행준위를 표시한다).

22.13 말단관리

워크스테이션과 PC가 출현하기전에는 말단들이 UNIX세계를 지배하였다. 그것들은 오늘날 도형출력이 요구되지 않는 장소들에서 여전히 사용되고 있다. 말단은 제공되는 직렬포구들중의 하나를 통해서든가 아니면 다중포구를 제공하는 별개의 직렬포구기판을 통하여 UNIX기계와 접속될수 있다. PC는 2개의 직렬포구(COM1과 COM2)를 가지고 있으며 말단이나 모뎀, 마우스를 접속하는데 사용될수 있다.

22.13.1 getty와 /etc/gettydefs

init는 시동스크립트(startup script)를 통하여 체계데몬을 시동시키는것외에 직접 말단을 조종한다. 그것은 말단에 접속된 모든 직렬포구들에 getty를 적재한다. System V상에서 매 말단은 /etc/gettydefs의 항목에 대응하는 /etc/inittab의 개별적인 항목을 요구한다. 아래에 2개의 일반적인 항목을 보여 준다.

```
C02:234:respawn:/sbin/getty tty02 m          말단에 련결되였다
C03:234:off:/sbin/getty tty03 m              말단에 련결되지 않았다
```

첫행을 번역하면 다음과 같다. 《실행준위 2, 3, 4를 위하여 두개의 인수 tty02와 m을 가지고 getty 프로그람을 실행시키시오.》 respawn설정은 말단이 항상 가능상태에 있도록 한다. 즉 getty는 사용자가 탈퇴하는 경우 그것이 제거될 때마다 다시 생성된다. 여기서 동작(action)이 off로 설정되기때문에 말단 tty03에서는 가입프롬프트를 볼수 없다.

init는 화면상에 login:프롬프트를 만들기 위하여 getty를 분기생성하여 실행시킨다(fork-execd). 그것은 또한 자기의 2번째 인수(m)를 /etc/gettydefs의 첫 마당과 정합한다. 이 파일에는 말단정의가 들어 있다.

```
l# B4800 HUPCL # B4800 CS8 SANE HUPCL TAB3 ECHOE IXANY #\r\nlogin: # m
m # B9600 HUPCL # B9600 CS8 SANE HUPCL TAB3 ECHOE IXANY #\r\nlogin: # n
n # B19200 HUPCL # B19200 CS8 SANE HUPCL TAB3 IXANY #\r\nlogin: # l
```

여기서 getty는 처음에는 m으로 시작된 2번째 행을 사용한다. 그것은 말단속도를 9600보(baud)로 설정하고 가입프롬프트를 만든다. 이 행의 마지막마당에는 또 다른 표식(n)이 있는데 getty는 그 행에서 통신이 되지 않으면(break가 수신되면) 다음것을 비교해 본다. 순환적인 탐색이 설정된다. 즉 처음에는 9600, 그다음은 19200, 그다음에는 4800, 그리고 다시 9600으로 설정된다. 만일 행의 첫 꼬리표가 마지막 꼬리표와 같으면 그러한 순환은 불가능할것이다.

일단 말단이 설정되였고 두 파일안에 항목들이 만들어 졌다면 init에 /etc/inittab을 다시 읽도록 요구해야 한다. 류사하게 /etc/gettydefs 안에 만들어 진 변화는 getty의 -c선택항목으로 검사해야 한다.

```
telinit q
getty -c /etc/gettydefs
```

Solaris는 gettydefs를 사용하지 않는다. 여기서 getty는 inittab안의 선택항목들과 함께 호출되는 ttymon이라는 포구감시프로그람에 대한 기호련결이다.

Linux

Linux는 여러개의 getty데몬들 즉 agetty, mingetty, uugetty를 가지고 있다. 또한 직렬포구에 련결된 모뎀들을 조작하기 위한 별개의 데몬 즉 mgetty도 사용한다. 그것들중 어느것도 gettydefs를 사용하지 않는다. 그것들은 서로 다른 구성파일을 사용하거나 몇개의 gettydefs형의 선택항목들과 함께 호출된다.

22.13.2 말단형의 설정

대부분의 체계들은 말단정의외에 말단의 형태들을 보관하기 위하여 개별적인 파일을 사용한다. 대체로 이 파일은 /etc/ttytype이다. 이 파일은 매 말단장치이름을 위한 행을 포함하며 2개의 마당 즉 말단형과 말단이름을 포함한다.

```
vt220  tty01
ansi   tty1a
dialup  tty2A
```

말단형은 간단히 말단특성들을 조종하는 자료기지인 /usr/lib/terminfo의 항목이다. 실례로 vt220속성들은 2진파일 /usr/lib/terminfo/v/vt220안에서 찾을수 있다. 원래 TERM변수는 /etc/ttytype를 읽어서 설정한다. 다른 말단특성들은 stty로 설정할수 있다.

주해 만일 어떤 말단이 다중포구기판를 통하여 접속되였다면 inittab안에 포구의 개수만큼 많은 항목이 있게 된다. 매 말단에 대하여 동작마당이 가능한가를 확인해야 하며 그렇지 않으면 그 말단은 작업할수 없다.

22.13.3 가상말단의 사용

만일 PC상에서 UNIX나 Linux를 사용하고 있다면 같은 기계상의 여러 말단들에 두번이상 가입할수 있다. [Alt][F2]나 [Alt][F3]을 사용하여 보면 그것을 알수 있다(화면상에서 login:프롬프트를 보게 될것이다). UNIX는 여러개의 가상말단을 지원하며 [Alt]건과 기능건을 함께 사용하여 매 가상말단을 호출할수 있다. 이때 가상말단들중 임의의것에 가입하여 같은 사용자이름이나 다른 사용자이름을 가지고 각각의 가입대화를 시작할수 있다.

사용자는 한 말단에서 프로그람을 편집하여 다른 말단상에서 그것들을 번역하거나 실행시킬수 있다. 그 기술은 아주 단순하다. 한 화면상에서 편집기(vi또는 emacs)를 프로그람과 함께 불러 낸다. 편집기에서 탈퇴하지 않고 편집된 변화를 보존한 다음 다른 창문으로 절환하여 콤파일지령이나 실행지령을 동작시킨다. 이 지령행은 오직 한번만 입력한다(Bourne쉘을 사용하지 않는다고 가정하고). 계속하여 !! 혹은 r(마지막지령을 반복하기 위하여)만을 사용하면 된다.

이와 같은 가상말단들을 사용하기 시작하면 창문환경에서보다 더 빨리 작업할수 있다는것을 알게 될것이다. 따라서 많은 사람들이 UNIX를 사용한다.

22.14 인쇄의 기초

인쇄기관리는 관리자과제의 중요한 부분이며 대규모설치들에서 종종 많은 시간이 걸리군 한다. UNIX는 여러개의 지령들을 가지는 완충보조체계(spooling subsystem)를 제공하는데 그 지령들은 순서대로 조작이 진행되는가를 확인한다. 이것은 인쇄대기렬의 관리, 인쇄기의 추가와 삭제, 보조체계의 시동과 중지, 지정된 인쇄기의 관리를 주관한다. 관리지령들은 SVR4와 Linux에서 좀 다르다.

사용자가 파일을 인쇄하기 위해 lp를 사용할 때 파일은 줄 지어 대기렬에 들어 간다. SVR4에서의 완충은 lpsched데몬에 의하여 수행된다. 이 데몬은 체계가 다중사용자방식으로 넘어 갈 때 rcn.d등록부안의 시작스크립트에 의하여 호출된다. 인쇄활동을 잠시 중지하고 lpsched가 기동하고 있는가를 먼저 검사하기 위하여 아래의 순서렬을 사용해야 한다.

```
ps -e | grep lpsched
```

만일 기동하고 있지 않다면 rc2.d 또는 rc3.d안에서 S*lp패턴(lpsched를 위하여)과 맞는 "S"스크립트를 찾아 볼수 있다.

```
ls /etc/rc2.d/S*lp
```

《lp》접미사는 한때 보편적이였던 행인쇄기(line printer)들을 가리키였는데 오늘날에는 레이자인쇄기와 잉크분사식인쇄기의 출현으로 하여 쓸모 없게 되였다. 그러나 그 접미사는 기초기술을 가지고 있는것으로 하여 변화시키지 않고 있다. 오늘날의 인쇄기들은 대체로 출력이 수많은 점배렬로 구성되는 **비트배렬장치**(bitmap device)이다. 600×600dpi의 해상도를 가지는 인쇄기는 수직, 수평방향으로 인치당 600점을 인쇄한다. UNIX체계는 그러한 인쇄기들을 조작하도록 설계되지 않았다.

이러한 현대적인쇄기들은 **페지서술언어**(page description language:PDL)로 형식화된 입력을 받아들인다. 오늘날 이 PDL선택항목들은 바로 두가지 즉 Adobe의 Postscript와 Hewlett Packard의 PCL로 제한되였다. Postscript는 괄호와 대괄호, 사선들의 집합으로 특징 지어 지는 프로그람작성언어이다. UNIX지령들은 이 형식의 출력을 할수 없다. 다시 말하여 전통적인 UNIX문자출력을 Postscript인쇄기는 전혀 받아 들이지 않는다.

두 기술을 조화롭게 결합시키기 위해서는 문자자료가 어떤 **대면부프로그람**(interface program)을 통과해야 한다. 이 대면부는 광고페지의 인쇄, 페지규격의 설정, 인쇄기에서 사용하는 조종코드의 지정과 같은 일부 선택항목들을 조종하는 쉘스크립트이다. 대면부는 파일을 최종적으로 인쇄하기 위하여 종종 Ghostscript(gs)나 netpr와 같은 외부프로그람을 호출한다.

주해

대면부프로그람은 Microsoft Windows의 인쇄구동프로그람과 같은것으로 생각할수 있으며 인쇄기를 설치할 때 지정해야 한다. 만일 대면부프로그람을 정확히 지정하지 않는다면 인쇄기가 제대로 동작하지 않을것이다.

Linux

lpd체계

Linux에서 완충은 lpd데몬에 의하여 수행되며 BSD체계를 사용한다. 그것이 기동하고 있는지를 검사하려면 다음과 같이 하면 된다.

```
ps ax | grep lpd
```

만일 기동하고 있지 않으면 rc2.d 또는 rc3.d에서 S*lpd패턴(lpd를 위하여)과 맞는 "S"스크립트를 찾아 보아야 한다.

```
ls /etc/rc.d/rc2.d/S*lpd
```

Linux는 Postscript출력을 발생시킬수 있는 몇개의 도구(gs, a2ps, enscript 등)를 가지고 있다. ASCII입력으로부터 Postscript를 발생시키기 위해서는 a2ps지령을 사용하며 그 반대변환에는 ps2ascii를 사용한다.

22.15 SVR4인쇄기의 관리

lp와 별도로 완충체계는 아래와 같은 주요요소들로 구성된다.

- `cancel`(작업을 취소)과 `lpstat`(인쇄대기렬을 감시)와 같은 사용자지령들. 사용자는 자기가 제출한 일감만을 취소할수 있다.

- lpadmin, lpsched, lpshut, accept, reject, enable, disable과 같은 관리자지령들

유감스럽게도 이 지령들에 대한 표준적인 위치는 없다. 그것들중 일부는 /usr/lib에서, 일부는 /usr/sbin에서 찾을수도 있다. 이 묶음에서 관리자지령은 lpadmin프로그람이다. 이 지령은 인쇄기를 추가 또는 삭제하고 그 구성을 변경하는데 사용된다. 몇개의 선택항목이 있는데 그것들은 다음과 같다.

- 인쇄기이름의 할당(-p)
- 클라스의 정의(-c)
- 대면부스크립트의 지정(-e와 -i)
- 인쇄기장치파일의 지정(-v)
- 인쇄기형태의 정의(-m)
- 기정인쇄기의 설정(-d)
- 인쇄기의 삭제(-x)

lpadmin은 인쇄출력의 최종수납자를 인쇄기라기보다는 오히려 어떤 목적지라고 간주한다. 이 목적지는 한대의 인쇄기일수도 있고 류사한 인쇄기들의 한 모임일수도 있다. 3개의 일감이 같은 모임에 떨어지면 lpsched는 그 클라스의 인쇄기들에 그 일감을 나누어 준다. 모든 인쇄기가 동작상태이라면 처음으로 해방되는 인쇄기에 자동적으로 일감을 돌린다.

lp는 모든 요청을 등록부 /var/spool/lp/request에 완충한다. lpadmin에서 정의된 모든 인쇄기들은 여기에 하나의 등록부를 가지고 있다. BSD체계와 달리 이 등록부는 lp가 -c선택항목과 함께 사용되지 않는 한 파일의 복사본을 잡아 두지 않는다. 만일 spool된후에 파일을 변화시킨다면 lp의 호출방법이 그 변화를 최종출력에 반영할것인가를 결정한다.

lp는 또한 lpsched의 2번째 실체가 시작되지 않도록 하기 위하여 자물쇠파일 /usr/spool/lp/SCHEDLOCK를 만든다. 만일 인쇄가 비정상적으로 완료된다면 그 파일은 삭제되지 않는다. 이 경우에는 lpsched를 다시 불러 내기전에 이 파일을 삭제하여야 한다.

22.15.1 인쇄기의 추가

이 선택항목들과 지령들의 의미는 체계에 인쇄기를 추가하는것으로 인차 리해된다. lpsched가 실행되고 있다면 먼저 lpshut(Solaris는 제외)를 가지고 그것을 정지시키고 그다음 lpadmin을 사용하여 인쇄기를 설치하여야 한다.

```
# lpshut
Print services stopped
# lpadmin -p pr1 -m epson -v /dev/lp0
```

이것은 epson형의 인쇄기에 이름 pr1을 할당하고 장치 /dev/lp0에 소속시킨다. 또한 그 인쇄기를 위한 대면부프로그람을 /usr/spool/lp/model로부터 /usr/spool/lp/interface(Solaris에서는 /etc/lp/interfaces)에로 복사한다. 표준대면부프로그람은 model등록부에 보관되며 설치프로쎄스는 interface(s)등록부에 적당한것을 복사한다. lp는 원래것이 아니라 복사된 프로그람을 사용한다.

대면부프로그람은 사실상 lpsched가 몇개의 인수를 가지고 호출하는 쉘스크립트이다. 이것들은 사용자이름, 복사본의 개수, 파일이름, 일감ID를 포함한다. 스크립트는 지정된 파일이름으로부터 입력을 얻어 인쇄기에 맞는 형식으로 만든다. 이 출력은 -v선택항목으로 인쇄기를 구성할 때 설정되였던 장치파일에로 다시 보내여 진다.

우에서는 단독인쇄기를 정의하였지만 -c선택항목을 가지고 모임에 속하는 인쇄기를 정의할수도 있다. 모임이 존재하지 않는다면 새롭게 만들어 진다.

```
lpadmin -p pr1 -m epson -v /dev/lp0 -c c1          pr1은 c1모임에 속한
```

-p선택항목은 인쇄기를 추가할 때와 그의 구성을 수정할 때 사용된다. 뿐만아니라 lp보조체계의 많은 지령들이 인쇄기를 지정하기 위하여 이 선택항목을 사용한다. 이제 lpsched를 시작하고 lp에게 요청수락개시를 지시한 다음 인쇄기를 가능상태로 만들어 보자.

```
# lpsched
Print services started.
# accept pr1
destination "pr1" now accepting requests
# enable pr1
Printer "pr1" now enabled          인쇄기는 현재 설치되여 인쇄할수 있다
```

아직 체계를 위한 기정인쇄기(default printer)가 정의되지 않았다. -d선택항목이 기정인쇄기를 설정한다.

```
lpadmin -d pr1          기정인쇄기로 pr1을 설정한다
lpadmin -d cla1         기정모임으로 cla1을 설정한다
```

제대로 동작하지 않는 대면부프로그람이 있다면 그것을 바꾸기 위하여 -m선택항목을 사용해야 한다. 대면부가 만족스럽다면 그때는 류사한 인쇄기를 추가할 때 현재 존재하는 인쇄기의 대면부프로그람을 복사하도록 lpadmin(-e와 함께)에 지시할수 있다.

```
lpadmin -ppr1 -m HPLaserJet          HPLaserjet의 대면부프로그람을 사용
lpadmin -ppr3 -e pr1                 pr1에서 사용된 대면부프로그람을 사용
```

-x선택항목은 인쇄기에 대기하고 있는 일감이 없다고 제기하는 체계로부터 그 인쇄기를 삭제한다.

```
lpadmin -x pr1          인쇄기 pr1을 제거한다
```

주해

기본인쇄기는 LPDEST변수에 의하여 처음 설정된다. 만일 정의되지 않았다면 lpadmin -d에 의하여 설정된 초기값이 사용된다. 정의된 기본인쇄기가 없다면 lp는 인쇄요청을 거절한다.

22.15.2 인쇄기와 일감상태얻기(lpstat)

lpstat는 인쇄기와 일감의 상태정보를 제공하는 여러개의 선택항목을 가진다. -r선택항목은 lpsched가 실행되고 있는가를 보여 준다.

```
# lpstat -r
scheduler is running
```

선택항목이 없으면 lpstat는 그 지령을 실행시킨 사용자가 제기한 모든 요청의 상태를 보여 준다. 관리자는 지정한 인쇄기에 대한 일감들의 목록을 얻을수도 있다.

```
# lpstat -ppr1          -ppr1을 사용할수도 있다
pr1-323  romeo    345670    Jan 10 13:26    on laser
```

```
pr1-324  romeo        3659        Jan 10 13:30
pr1-325  juliet      23678        Jan 10 13:40
```

일감들은 또한 사용자(-u)에 의해 렬거될수도 있으며 -t선택항목으로 정보목록을 얻을수도 있다.

```
# lpstat -t
scheduler is running
system default destination: pr1
device for pr1: /dev/lp0
pr1 accepting requests since Fri Dec 12 10:06:04 1997
printer pr1 waiting for auto-retry. available.
    stopped with printer fault
pr1-1               henry         1897       Dec 12 10:37
```

주해

일정작성기(scheduler)는 lpshut로 정지되고 lpsched로 시작된다. lpsched가 다시 시작되면 중지되였던 모든 일감들이 처음부터 인쇄를 시작한다.

22.15.3 완충에 대한 조종(accept와 reject)

몇개의 지령들이 완충(spooling)동작을 조종한다. 자료전송량이 높거나 통과량이 없는 인쇄기에 의하여 재정렬해야 한다. 관리자는 완충기로 들어 오는 입력을 조종하고 한 인쇄기로부터 다른 인쇄기로 일감들을 넘길수 있어야 한다.

어떤 일감을 취소하기 위해서는 cancel지령을 사용한다. 그렇지만 완충기의 통과량이 높을 때는 완충기자체에로의 입력을 조종하기 위하여 accept와 reject를 사용해야 한다. reject는 완충을 억제하고 그 리유를 지정하기 위하여 -r선택항목을 사용한다면 거절된 인쇄기상에서 인쇄를 시도하던 사용자는 그 리유를 통보문으로서 보게 될것이다.

```
# reject -r"Spooler very busy" laser epson
destination "laser" will no longer accept requests
destination "epson" will no longer accept requests
```

accept지령은 완충을 허락하기 위하여 사용된다. 새로운 인쇄기가 체계에 추가될 때마다 이 지령이 실행되여야 한다. 그러나 accept와 reject는 둘 다 절박한 일감의 상태에 영향을 미칠수 없다.

22.15.4 enable과 disable

accept는 완충을 허용하지만 일감이 인쇄될것이라는것을 자동적으로 암시하지는 않는다. 그렇게 하도록 하기 위해서는 목적인쇄기가 가능상태로 되여야 하며 그것은 인쇄기가 추가될 때 일단 수행된다. 그러나 특수한 인쇄기가 꺼져야 할 경우가 있다. 실례로 인쇄종이가 막혀 나오지 못하는 때를 들수 있다. disable지령은 그 인쇄기의 인쇄동작을 정지시킨다.

```
disable -r"Jamming of paper; just a minute" lwriter
```

reject와 달리 disable은 완충기에 들어 가는 일감을 막지는 않고 현재일감의 인쇄를 중지시킨다. 불가능상태로 된 인쇄기에 공급되였던 모든 일감은 오직 그 인쇄기가 enable지령에 의하여 가능상태로 되였을 때에만 인쇄될수 있다.

22.15.5 일감을 다른 인쇄기에 보내기(lpmove)

이와 같이 인쇄기가 불가능으로 되였을 때는 lpmove를 사용하여 일감들을 다른 인쇄기에로 옮길수 있다. 지령은 두가지 방법으로 사용될수 있다.

```
lpmove pr1-321 pr1-322 laser
```
두개 일감들은 laster에 옮겨 진다

```
lpmove pr1 laser
```
pr1의 모든 일감들은 laster에 옮겨 진다

2번째의 경우 lpmove는 원천인쇄기상에 기정적인 reject를 불러 낸다. 여기서 그 지령은 인쇄기 pr1은 더이상 요구를 받아 들일수 없고 그의 모든 일감들이 옮겨 진다는것을 의미한다.

Linux

Linux인쇄기의 관리

Linux는 낡은 버클리인쇄체계를 사용한다. 그것은 좀 미숙하며 AT&T체계만큼 종합적이지 못하다. 여기서 일감들은 lpr지령에 의하여 인쇄되며 lpd데몬을 사용한다. SVR4의것과 비교하여 보면 인쇄체계는 더 적은 지령들을 가지고 있다.

- lprm(일감취소)과 lpq(인쇄대기렬감시)와 같은 사용자지령들
- 모든 인쇄기를 감시하는 관리자지령 lpc와 SVR4안의 개별적인 지령들에 의하여 수행되는 완충기능

AT&T의 lpadmin과 달리 Linux는 인쇄기설치를 위한 지령행도구를 가지지 않는다. Linux체계들은 인쇄기를 구성하기 위하여 차림표로 이루어 진 쏘프트웨어를 제공한다. 앞단(front-end)들을 통하여 제공되는 대부분의 입력들은 최종적으로 본문파일 /etc/printcap에서 끝난다. 이 자료기지는 이름, 대면부스크립트, spool등록부, 장치파일 등과 같은 모든 인쇄기속성을 보관한다. 이 파일은 직접 편집할수 있으나 그 파일에서 사용된 변수나 코드들을 잘 아는 경우에만 시도할수 있다.

사용자가 인쇄완충기에 일감을 맡길 때 lpr는 그 파일이 배치될 등록부를 /etc/ printcap에서 찾는다. 파일의 복사본이 그 위치에 배치된다. 이 대기렬은 인쇄기데몬 lpd에 의하여 감시된다. lpd는 인쇄대기중의 일감을 발견하면 자기의 또 다른 복사본을 생성한다. 그다음 새로운 lpd프로쎄스가 그 일감을 인쇄하기 위하여 /etc/printcap 에 지정된 대면부프로그람을 불러 낸다.

이 체계안의 거의 모든 지령들은 인쇄기를 지정하기 위해 사용되는 -P선택항목(System V에서는 -p)을 리해한다. 이 선택항목이 사용되지 않는다면 그 지령은 체계에 정의된 기정인쇄기를 리용한다.

lpq와 lprm: 인쇄기와 일감상태를 얻기

lpq는 인쇄대기렬을 보여 준다. 인쇄기선택항목 -P와 함께 사용된다면 그 인쇄기의 대기렬을 현시한다. 기정적으로 lpq는 기본인쇄기상의 일감들의 상태를 현시한다.

```
# lpq
lp is ready and printing
Rank     Owner    Job  Files                          Total Size
active   sumit    48   (standard input)               1972 bytes
1st      sumit    49   searchlist.html                61061 bytes
2nd      henry    50   (standard input)               17746 bytes
```

lpq가 active를 첫 일감으로 보여 준다는 사실은 lpd데몬이 기동한다는것을 의미한다. 그렇지 않으면 1st일감을 보여 주었을것이다. 일감번호는 3번째 렬에서 보여 준다. lpstat와 달리 lpq는 관흐름으로부터 입력을 받았을 때에는 표준입력을, 그렇지 않은 경우에는 파일이름을 보여 준다.

-l선택항목을 가지고 lpq를 사용하면 인쇄대기렬상태의 종합적인 출력을 얻을수 있다.

```
# lpq -l
```

```
lp is ready and printing

sumit: active            [job 048uranus]
       (standard input)  1972 bytes
sumit: 1st               [job 049uranus]
        searchlist.html  61061 bytes
```

lprm지령은 인쇄대기렬로부터 일감을 삭제한다. 권한 없는 사용자도 이 지령을 사용할수 있으나 그 지령은 자기에게 소유된 일감만으로 작업한다. 그렇지만 상급사용자가 인수로써 -를 가지고 이 지령을 사용하면 모든 일감들이 대기렬에서 해제된다. lprm은 어떤 인쇄기에 관계되는 모든 일감들이나 어떤 사용자에 소유된 일감들을 제거할수도 있다.

```
lprm -            대기렬에 있는 모든 일감들을 제거한다
lprm -Php1        hp1인쇄기에 있는 모든 일감들을 제거한다
lprm henry        henry가 가지고 있는 모든 일감들을 제거한다
```

주해

기정인쇄기는 PRINTER변수에 의하여 처음으로 설정된다. 만일 정의되지 않았다면 /etc/printcap에서 설정된 기정값이 리용된다. 거기에 정의된 기정인쇄기가 없다면 lpr는 인쇄요구를 거절한다.

/etc/printcap: 기본자료기지

BSD lpd체계는 모든 인쇄기명세들을 보관하기 위한 자료기지로서 /etc/printcap을 사용한다. 이 파일은 말단정의를 보관하는 /etc/termcap과 류사한 애매한 구조를 가지고 있다. :s와 #s를 가진 일부 변수들이 때때로 달리 해석될수 있다. 자기가 하고 있는것을 확인하지 않는 한 앞단쏘프트웨어(Red Hat에서의 printtool과 같은)가 적어도 처음에 이 파일을 수정하게 하는것이 더 좋다.

printcap는 부분들로 나누어 지며 매 부분은 인쇄기정의를 보관한다. 그 정의는 인쇄기의 이름으로부터 시작하며 마지막에 |에 의해 구분된다. HP Deskjet 500인쇄기에 대한 간단한 printcap항목을 보기로 하자.

```
lp|lwriter|cdj500-a4-auto-mono-300|cdj500 a4 auto mono 300:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/cdj500-a4-auto-mono-300:\
        :lf=/var/spool/lpd/cdj500-a4-auto-mono-300/log:\
        :af=/var/spool/lpd/cdj500-a4-auto-mono-300/acct:\
        :if=/var/lib/apsfilter/bin/cdj500-a4-auto-mono-300:\
        :la@:mx#0:\
        :tr=:cl:sh:sf:
```

여기서 인쇄기는 lp, lwriter, cdj500-a4-auto-mono-300이라는 이름을 가지고 있다. -p선택항목을 가지고 이 이름들중 어느 하나를 사용할수 있다. 다른 행들은 각각 인쇄속성을 포함하며 두점(:)으로 구분된다. printcap정의에서 마지막행을 제외하고 모든 새로운 행들은 \로 끝나야 한다.

모든 인쇄속성은 변수=값의 형식을 취한다. printcap는 인쇄기장치이름, spool등록부, 페지규격, 려파프로그람과 같은것들을 조종하는 몇개의 변수들을 포함한다. 여기서 lp는 인쇄기이름과 변수이며 사용되는 문맥에서 그 의미에 대하여 오유를 만들지 말아야 한다. 여기서 lp변수는 /dev/lp0 즉 병렬포구에 인쇄기를 설정한다. Linux체계들은 3개의 병렬포구를 사용하며(lp1, lp2 등) 따라서 장치이름을 정확히 선택하였는가를 확인해야 한다.

spool등록부(sd)는 lpr가 인쇄를 위해 파일들을 떨구는 /var/spool/lpd안에 있는 등록부이다. 등록부이름으로써 인쇄기이름을 사용한다. 매 파일을 위하여 lpr는 spool등록부안에 cf와 df접두사를 가진 2개의 파일을 만든다. df파일은 파일의 려파되지 않은 자료를 포함하며 cf파일은 조종정보를 포함한다.

if는 입력려파기를 설정한다. 여기서는 /var/lib/apsfilter/bin안에 보관된 대면부쉘스크립트를 가리킨다. 입력려파기는 보통 X Window기반의 도형프로그람(gs, Ghostscript프로그

람과 같은)을 호출한다. 우리는 다른 인쇄속성은 무시할것이다.

spool등록부는 두개의 일감이 동시에 한 인쇄기에 접근하는것을 막는 자물쇠파일을 포함한다. 자물쇠파일(lock file)은 cf값을 번호로 가지는 현재능동인 일감과 lpd데몬의 PID를 포함한다. 이 파일은 인쇄가 끝난후에 제외되며 다음일감을 위하여 새로운 파일이 만들어 진다. 만일 인쇄도중에 전원고장이 일어 나면 이 파일을 수동으로 삭제해야 한다.

printcap파일을 변화시킨후에는 lpd데몬을 다시 시작해야 한다. init.d안의 lpd스크립트는 start, stop과 함께 reload와 restart인수도 지원한다. 사용자는 이 스크립트든가 그에 련결된 rc스크립트를 사용할수 있다.

```
/etc/rc.d/init.d/lpd restart
/etc/rc.d/rc3.d/s60lpd restart
```

lpc: 인쇄기조종

lpc지령은 모든 인쇄기조종기능을 조작한다. 지령 그자체만으로 사용될 때는 lpc>프롬프트를 만든다. 사용자는 이 프롬프트로부터 모든 내부지령을 입력할수 있다. 그것들은 다음과 같은 목적으로 사용된다.

- 완충을 가능/불가능상태로 하는것(enable과 disable)
- 인쇄기를 가능/불가능상태로 하는것(start와 stop)
- 모든것을 가능/불가능상태로 하는것(up과 down)
- 인쇄기데몬을 조종하는것(abort와 restart)
- 완충상태와 인쇄상태를 현시하는것(status)
- 대기렬의 꼭대기로 일감을 옮기는것(topq)

완전한 기능의 목록을 현시하기 위해서는 lpc>프롬프트에 help를 입력할수 있다. lpc는 이 지령들을 자기의 인수로 하여 비대화방식에서 실행될수도 있다. 이 기능의 일부는 AT&T체계에서 공통이며 따라서 그것들을 여기서는 간단히 서술하도록 한다.

topq를 제외한 이 모든 지령들은 선택적인 인수로써 인쇄기이름을 사용한다. 만일 인쇄기이름이 지원되지 않는다면 기본인쇄기가 사용된다. System V사용자들은 lpc가 완충을 조종하기 위하여 enable및 disable기능을 사용한다는데 주의를 돌릴 필요가 있다. 우리는 쉘의 지령행으로부터 그것들중 하나를 사용하며 lpc>프롬프트에서 다른것을 사용한다.

```
# lpc enable lwriter
pc> disable lwriter
```

start와 stop지령들은 인쇄동작을 조종하지만(AT&T에서 enable, disable과 같은) 추가적으로 lpd의 시작과 정지를 조종한다. 줄세우기와 인쇄를 포함하는 모든것을 조종하기 위해서는 up과 down지령을 사용해야 한다. AT&T체계의 reject지령과 같이 down지령도 인수로써 통보문을 받아 들인다.

```
up lwriter
down lwriter "Printer is down"
```

관리자는 제한된 방법으로 일감의 우선권을 변경시킬수 있다. 즉 그것을 대기렬의 꼭대기로만 옮길수 있으며 그밖의 다른 곳으로는 옮길수 없다.

```
topq lwriter 52
topq lwriter henry
```

status지령은 인쇄체계가 놓인 현재상태를 보여 준다. 그 지령은 임의의 사용자에 의하여 사용될수 있다.

```
lpc> status lwriter
lwriter:
      queuing is enabled
      printing is enabled
      no entries
      printer idle
```

참고

현재일감의 인쇄를 즉시 중지하려면 lpc의 abort지령을 사용하시오.

요 약

뿌리사용자등록자리는 체계관리자를 위해 사용된다. su지령은 뿌리의 권한을 요구하는 임의의 사용자등록자리로부터 호출될수도 있다. 관리지령의 대부분은 /sbin과 /usr/sbin에 상주한다. PATH는 현재등록부를 포함하지 않는다.

관리자는 파일의 속성을 변경시킬수 있으며 임의의 프로쎄스를 제거하고 임의의 사용자통과암호를 바꿀수 있다. 그는 체계날자를 설정하고 모든 사용자들의 주소를 동시에 지정할수 있으며(wall) 그들에게 자기들의 계약을 상기시킬수 있으며(ccalendar) 파일크기의 한계를 설정(ulimit)할수 있다.

사용자가 추가(useradd), 변경(usermod)될수 있으며 체계로부터 삭제(userdel)될수 있다. 사용자의 상세정보는 파일 /etc/passwd, /etc/shadow, /etc/group안에서 관리된다. 통과암호는 shadow안에 부호화된 방식으로 보관된다. 일부 체계의 사용자들은 자기들이 가입할 쉘을 자체로 바꿀수 있다(chsh).

파일의 사용자ID설정비트(SUID)는 사용자가 소유자의 권한을 실행과정에 일시적으로 얻는 프로그람들을 실행시킬수 있게 한다. 이것은 사용자로 하여금 직접적으로가 아니라 이 지령들을 사용하여 체계파일을 변경시키거나 체계등록부안에 써넣을수 있게 하여 준다.

보통 파일상에서의 점착비트는 그 파일을 교체구역안에 영구적으로 보존한다. 등록부상에서 사용될 때에는 어느 한 그룹에 속한 사용자들이 자기들의 파일들을 만들고 삭제할수 있게 한다(다른 그룹성원들의 파일은 조작할수 없다). /tmp와 /var/tmp와 같은 대부분의 UNIX체계등록부들이 이 비트를 설정하고 있는데 그 리유는 사용자들이 보안을 위반하지 않고 이 등록부에 파일을 써넣을수 있게 하기 위한것이다.

제한쉘은 사용자가 자기 등록부를 변경시키거나 다른 등록부에 위치한 지령들을 실행시키거나 지어는 파일들을 만드는것도 허용하지 않는다. 접근할수 있게 하려면 체계지령들이 사용자등록부에 배치된 파일들에 련결되여야 한다.

init프로쎄스는 지정된 실행준위에서 체계를 관리하며 사용자가 가입할수 있도록 모든 말단에 getty을 생성한다. init는 /etc/inittab로부터 지시를 읽으며 지정된 실행준위와 관계되는 체계데몬(프로쎄스)들을 실행시키는것을 담당한다. 실행준위 2 또는 3은 보통의 다중사용자조작을 표현하며 1 또는 s는 단일사용자조작을 표현한다.

shutdown프로그람은 기계를 끄기전에 프로쎄스들을 제거하고 파일체계를 내리운다. init도 기계를 끄기 위하여 실행준위를 인수로 하여 사용될수 있다(0 또는 6). Linux에서는 [ctrl][Alt][Del]순서렬도 기계를 정지시킨다.

플로피들은 format나 fdformat로 초기화된다. dd는 문자장치를 사용하며 디스케트와 테프들을 복사하는데 쓰인다. UNIX는 DOS디스케트를 조작하기 위한 지령묶음을 제공한다. 그것들은 System V에서는 문자렬 dos로 시작하며 Linux에서는 m으로 시작한다.

cpio는 파일들을 하나의 보존파일로 묶기 위하여 표준입력으로부터 그 파일들의 목록을 얻는다. 그것은 관흐름에서 find와 함께 사용하는것이 아주 적절하다. 주요선택항목들을 사용하는데 매체에로의 복사는 -o, 매체로부터의 재보관은 -i, 보존파일의 현시는 -it를 사용한다. 그것은 원래 -u선택항목이 지정되지 않는 한 더 낡은 파일을 덧쓰지 않는다. 한 그룹의 파일들을 보관하기 위해서는 통용기호들도 사용될수 있다.

등록부나무를 여벌복사하는데는 tar가 더 적절하다. 그것도 주요선택항목을 사용한다. 즉 매체에로의 복사는 -c, 그로부터의 재보관은 -x, 보존파일의 현시는 -t를 사용한다. 보존파일에 추가(-r와 -u)할수 있고 같은 파일의 여러 판본을 보관할수도 있다. GNU tar는 보존동작(archive)에 압축(-z와 -Z)을 추가한다.

du지령은 모든 사용자의 홈등록부나무의 리용정형을 통보한다. 관리자는 오래동안 사용되지 않은 파일들의 위치를 알아 내기 위하여 find를 사용한다. find의 능력은 xargs와 함께 사용할 때 더 강화되며 그것은 파일이 발견될 때마다가 아니라 한번 혹은 지정된 회수만큼 지령을 실행시킨다.

보안을 실시함에 있어서 관리자는 passwd지령으로 통과암호로화를 실현하여 사용자들이 일정한 시간이 경과한후에는 자기의 통과암호를 바꾸게 할수 있다. 사용자등록자리를 잠글수 있고(-l) 사용자가 첫 가입시에 자기의 통과암호를 바꾸도록 할수 있다(-f).

체계프로쎄스들은 /etc안의 스크립트와 그 보조등록부들의 다른 스크립트들로부터 시작된다. 지정된 실행준위에서 실행될 스크립트들은 개별적인 등록부들에서 관리된다. K스크립트들은 프로쎄스들을 제거하는데 쓰이며 S스크립트들은 그것들을 시작시키는데 리용된다. 이 모든 스크립트들은 /etc/init.d안에 격납되여 있는 스크립트들에 련결되여 있다.

말단정의는 getty에 의하여 /etc/gettydefs로부터 접근된다. 그 파일은 getty가 파라메터묶음들을 비교하여 보도록 구성될수 있다. 기본말단형들은 많은 체계들에서 /etc/ttytype안에 보관된다. PC상에서 가상말단을 리용하여 여러개의 가입대화를 설정할수 있고 그우에서 서로 다른 일감들을 실행시킬수 있으며 각각의 개별적인 현시기를 가질수 있다.

System V상에서 lpsched데몬은 인쇄대기렬을 감시하고 동작시킨다. lpadmin은 인쇄기의 추가, 수정, 기정인쇄기설정에 리용된다. lpstat는 인쇄요구의 상태를 현시하며 accept는 일감을 받아 들이기 위하여 인쇄대기렬을 준비시키며 reject는 완충을 완전히 금지시킨다. 인쇄를 위해서는 인쇄기가 가능상태(enable)여야 하지만 때로는 불가능상태(disable)가 되여야 할 때도 있다. 대기렬안의 일감은 또 다른 인쇄기에로 옮겨 질수 있다(lpmove).

Linux는 BSD인쇄체계를 사용한다. lpd는 인쇄대기렬을 감시하는 인쇄기데몬이다. lpq는 대기렬의 상태를 현시하며 lprm은 대기렬로부터 일감을 삭제한다. 인쇄기특성은 /etc/printcap안에 보관된다. 인쇄와 완충조종은 lpc를 가지고 수행한다. 인쇄기를 가능 및 불가능하게 할수 있고(start와 stop) 완충을 설정하거나 해제할수 있다(enable과 disable). 일감을 대기렬의 선두로 옮겨 놓을수도 있다(topq).

UNIX인쇄프로그람들은 대면부프로그람 즉 인쇄기의 특성에 대응되게 자료를 형식화하는 려파기에로 인쇄자료를 통과시킨다. 려파기는 종종 실제적인 인쇄를 수행하는 외부프로그람(external program)을 호출한다.

시험문제

1. 관리자의 지령들은 어디에 위치하는가?
2. calendar지령의 동작은 상급사용자에 의하여 호출될 때 어떻게 변화되는가?
3. 통과암호는 어디에 보관되는가?
4. 관리자는 어떻게 모든 사용자들에게 동시에 주소를 지정하는가?
5. 사용자ID 212를 가진 사용자 john을 만들고 dialout그룹에 배속시키시오. john은 쉘로써 bash를 사용하며 /home등록부에 위치할것이다.
6. 뿌리등록자리를 사용하지 않고 자기의 쉘을 어떻게 바꿀수 있는가?
7. UNIX핵심부의 이름은 무엇인가?
8. 체계의 기정실행준위를 어떻게 결정하는가?
9. Linux에서 끄기절차를 초기화하는 가장 빠른 방법은 무엇인가?
10. 체계를 즉시에 끄기 위해서는 shutdown을 어떻게 사용하겠는가?
11. init의 두가지 중요한 역할은 무엇인가?
12. 실행준위 0과 6사이의 차이점은 무엇인가?
13. 체계가 단일사용자방식에 있을 때 파일을 인쇄할수 있는가?
14. /etc/inittab를 변경시킨후 인차 무엇을 해야 하는가?

련습문제

1. passwd를 상급사용자가 호출했을 때 그 지령의 동작은 어떻게 변하는가?
2. /etc/passwd의 첫 마당을 읽어 내는것으로써 어느 변수가 설정되는가?
3. telnet는 때때로 뿌리가입을 허락하지 않는다. 그때 원격주콤퓨터(remote host)상에서 뿌리에 의해서만 쓰기가능한 파일을 변화시키려면 어떻게 하겠는가?
4. 왜 통과암호부호화가 /etc/passwd로부터 /etc/shadow로 옮겨 졌는가?
5. passwd지령을 사용하지 않고 사용자 kathy가 자기의 등록자리로 가입하는것을 막으려면 어떻게 할수 있는가?
6. 가입후에 사용자가 등록부들을 변경시킬수 없거나 자기의 홈등록부안에 파일을 만들수 없다. 그것은 어떻게 일어 날수 있는가?
7. 파일이 쓰기허가권을 가지고 있지 않는데도 사용자가 어떻게 passwd를 가지고 /etc/shadow를 갱신할수 있는가?
8. 목록의 허가권마당에 문자 t가 보였다. 이 t는 무엇을 가리키는가?
9. 같은 등록부에 써넣기를 수행하며 다른 사용자의 파일은 지울수 없는 사용자그룹을 어떻게 준비할것인가?

10. 뿌리와 같은 권한을 가진 또 다른 사용자를 어떻게 생성할수 있는가?

11. 아래의 지령은 무엇을 하는가?

```
find / -perm -4000 -mount -print
```

12. 실행준위 1은 무엇인가? 이 상태에서 수행할수 없는 4가지 동작을 말해 보시오.

13. 오전 9시부터 저녁 10시사이에 매 시간 빈 디스크공간을 감시하기 위해서는 체계관리자가 어떻게 준비하여야 하는가?

14. DOS의 DISKCOPY지령을 실현하기 위한 쉘스크립트를 쓰시오.

15. 모든 HTML파일들을 어떻게 DOS플로피에 복사하는가?
    (1) SVR4에서 (2)Linux에서

16. 오늘 작업한 모든 파일들을 여벌복사하려고 한다. 가입후에 먼저 무엇을 해야 하며 cpio를 가지고 그것들을 어떻게 여벌복사하는가?

17. tar로 등록부구조 /home/local/bin을 보존하고 gzip를 가지고 그것을 디스크파일로 압축하시오.

18. 원격기계 jupiter상에서 tar를 가지고 등록부나무 htmldoc를 보존하여 자기의 국부하드디스크상에 보존파일을 만드시오. 그 지령이 언제 동작하지 않는가?

19. 통과암호로화정보는 어디에 보관되는가? 4주마다 사용자가 자기의 통과암호를 바꾸도록 하려면 어떻게 해야 하는가?

20. passwd지령을 사용하여 kathy가 자기의 등록자리를 사용하지 못하도록 하시오.

21. 시작스크립트와 제거스크립트란 무엇인가?

22. 말단에 가입프롬프트가 보이지 않는 경우 처음 검사해야 할것은 무엇인가?

23. SVR4체계에서 속도와 말단설정이 어디에 보존되는가? 말단이 접속된 곳과 속도를 어떻게 확인할것인가?

24. 체계가 SVR4를 사용하는지, Linux를 사용하는지에 관계없이 인쇄기데몬이 실행되고 있는가를 보여주는 순서렬를 고안하라.

25. 대면부스크립트란 무엇인가?

26. 인쇄될 자료가 spool등록부에 들어 있는가? 그렇지 않다면 어디에 있는가?

27. lpsched가 가동하고 있어도 현재인쇄기 laser1상에서 인쇄를 할수 없다. 먼저 무슨 지령을 써야 하는가?

28. SVR4에서 인쇄에 적용되는 항목인 accept와 enable의 차이점은 무엇인가?

제 23 장. TCP/IP망관리

이 장에서는 망관리자의 시점에서 TCP/IP를 다시 설명한다. 단독기계의 시대는 끝났다. 오늘날 사용자들은 응용프로그람까지도 망을 통하여 실행시킨다. 때문에 망관리는 관리활동의 필수적이며 전문화된 구성요소로 되였다. 대규모체계의 설치에서 망관리자는 흔히 체계관리자와 구별된다.

이 장에서는 먼저 TCP/IP리론을 설명한다. 간단한 TCP/IP망을 설치하고 다른 망의 주콤퓨터(host)에 접근하기 위해 그것을 구성하는 방법을 설명한다. 그리고 인터네트데몬을 가지고 기초적인 인터네트봉사를 설정하며 전화회선을 리용하여 인터네트에 접속하는 방법도 설명한다. 이 장에서는 또한 TCP/IP계렬의 마지막 2개의 기능인 PPP와 NFS도 설명한다. 마지막으로 DNS, SMTP, 전자우편, HTTP와 같이 망에 의해 사용되는 강력한 봉사들을 설명한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- TCP/IP망의 기능과 IP주소화체계를 리해한다(23.1).
- 망대면부기판을 설치하고 ifconfig로 구성한다(23.2, 23.3).
- ping을 리용하여 망의 고장을 진단한다(23.4).
- TCP/IP의 경로조종(routing)방법을 배우며 route를 리용하여 경로를 만들어 낸다(23.5).
- netstat를 리용하여 망상태를 현시한다(23.6).
- ftp와 telnet와 같은 기본적인 TCP/IP봉사를 조작하기 위한 inetd의 역할을 배운다(23.7).
- 점대점규약의 특성을 리해한다(23.8).
- dip와 chat를 써서 자기의 Linux기계를 인터네트에 련결한다(23.9).
- PPP상에서의 인증에 사용되는 두가지 중요한 통신규약 즉 PAP와 CHAP를 리해한다(23.10).
- NFS(Network File System)의 기능과 구성을 배운다(23.11).

23.1 TCP/IP와 주소화체계

TCP/IP는 계층화된 제품이며 여러개의 론리적계층들이 그의 여러가지 기능을 처리한다. 아래의 두가지 통신규약이 쓰이고 있다.

- **전송조종규약(Transmission Control Protocol:TCP)** - TCP는 전송의 믿음성을 확인하는것을 담당하며 강력한 오유검출 및 회복기능을 가지고 있다. 그것은 응용프로그람으로부터 받은 자료를 토막(파케트)으로 나누고 상대방이 옳바른 순서대로 재조립할수 있도록 렬번호(sequence number)와 검사합(checksum)을 붙인다. 이 파케트는 원천 및 목적포구번호도 포함한다. 만일 상대측에 손상된 토막이 수신되였다면 그 토막을 다시 전송한다. TCP는 또한 접속형이다. 즉 그것은 먼저 상대방과의 맞잡기(handshaking)시험을 시도하는것으로 원격체계가 자료를 교환할 준비가 되였는가를 확인한다. 이 맞잡기가 성공하였을 때 **접속**(connection)이 수립되며 그때 TCP는 자료전송을 시작한다. 대부분의 응용프로그람들이 TCP를 사용하지만 일부 응용프로그람(RealVideo와 같은)들은 이 간접조작시간(overhead)을 따를수 없으며 그대신 UDP와 같은 규약을 사용한다. UDP는 믿음성을 담보하지 못하며 접속지향도 아니다.

- **인터네트통신규약(Internet Protocol:IP)** - TCP층을 통과한 파케트는 IP층으로 옮겨 간다. IP는 파케트가 가야 할 곳과 떠나온 곳을 결정한다. 그것은 TCP토막에 원천 및 목적IP주소를 제공하며 또한 파케트를 정확한 목적지로 전달하는데서 중대한 역할을 한다. 파케트가 같은 망안의 어떤 주콤퓨터에로 향한다면 IP는 그 망으로 직접 파케트를 보낸다. 만일 그것이 다른 망의 주콤퓨터에로 향한다면 IP는 그 파케트를 가장 가까운 경로기에로 보낸다(이 경로기가 목적지에 가장 가까운 경로기라고 보고).

TCP와 IP가 노는 두가지 구별되는 역할에 주의하여야 한다. TCP는 포구번호 즉 접속되여야 할 응용프로그람을 지정한다. IP는 파케트가 어느 기계로 가며 어디서부터 오는가를 알게 한다.

TCP/IP망을 취급하기전에 관리자는 망에서 기계들이 주소화되는 방법에 대하여 알아야 한다. 매 기계는 망대면부기판을 가지고 있으며 그것은 다른 기계들의 대응하는 기판들과 선로를 통하여 련결된다. 주콤퓨터들사이의 모든 통신은 보통 이 망대면부들만을 통하여 수립된다.

모든 이써네트망기판들은 하드웨어제작자에 의해 기판안에 48bit의 물리적인 주소코드를 가지고 있다. 이 주소를 **MAC주소**(Media Access Control) 또는 **이써네트주소**(Ethernet address)라고 한다. 그것은 두점으로 구분된 6개의 16진수묶음으로 구성되며 대표적으로 다음과 같은것을 들수 있다.

```
00:00:E8:2E:47:0C
```

이 주소와 별도로 TCP/IP는 또한 **IP주소**(IP address) 즉 일부 TCP/IP도구들에서 자기가 사용한 론리적인 쏘프트웨어주소(인터네트주소)를 리해한다. 모든 주콤퓨터이름들이 대응하는 IP주소로 변환된다할지라도 TCP/IP는 최종적으로 주소를 그 주콤퓨터의 대응하는 MAC주소로 바꾸어야 한다.

또한 이 IP주소의 할당을 통제하는 몇개의 규칙이 있다. 다음의 32bit주소를 보자. 그것은 편리상 4개의 8비트들로 쪼개놓았다.

```
11000000 10101000 00000001 10100000
```

이 2진형식은 10진수 192.168.1.160으로 변환된다. 이 주소는 두개의 부분 즉 **망주소**(network address)와 **주콤퓨터주소**(host address)로 구성된다. 매 8비트들의 최대값은 255를 가질수 있다. 망주소는 같은 망안의 모든 주콤퓨터들에서는 같으며 왼쪽으로부터 1~3개의 8비트들을 사용할수 있다. 나머지8비트들은 주콤퓨터가 사용한다. 여기서 192.168.1은 망주소이다. 협약에 의하여 주콤퓨터부를 0으로 채우며 따라서 실제적으로는 192.168.1.0으로 읽어 진다.

망구성요소로 들어 가는 8비트들의 개수는 **부분망마스크**(subnet mask)에 의하여 결정된다. 이것은 또 련속적인 4개의 8비트들이며 망에 해당한 주소의 매 비트들은 1로 설정된다. 우의 망은 부분망마스크로서 255.255.255.0을 가진다. 이 마스크에 대하여 255로 설정될 8비트들의 개수는 일정한 규칙에 의해 결정된다. TCP/IP는 파케트가 현재의 망에 속하는가를 결정하는데 이 마스크를 사용한다. 후에 이 마스크에 대한 설명을 더 하겠다.

인터네트상에서 주소는 3개의 **클라스**중 하나에 속한다. 망주소로 예약된 8비트들의 개수와 첫 8비트들의 값이 그 망이 속하는 클라스를 결정한다. 3개의 클라스를 표 23-1에 보여 준다.

표 23-1. 망클라스와 예약주소

망클라스	첫 8비트의 값	부분망마스크	인트라네트를 위한 망주소
A	1 - 126	255.0.0.0	10.0.0.0 - 10.255.255.255
B	128 - 191	255.255.0.0	172.16.0.0 - 172.31.255.255
C	192 - 223	255.255.255.0	192.168.0.0 - 192.168.255.255

이 첫 8비트를 보면 망의 형태를 쉽게 말할수 있다. 실례로 148.27.3.12는 B클라스주소이며 192.142.3.67은 C클라스주소이다. 148.27.0.0과 192.142.3.0이 망주소이다.

B클라스망을 가질 때에는 그안에 65534개의 주콤퓨터를 가질수 있다. 3번째 8비트까지 리용해서 망을 정의한다면 126개의 주콤퓨터를 가질수 있는데 이것은 요구보다 더 적어 질수 있다. TCP/IP는 또한 보통 주콤퓨터주소의 구성요소를 형성하는 8비트들로부터 비트들을 빌려 오는것을 허용한다. 그러므로 만일 우의 B클라스망에서 3번째 8비트로부터 첫 두개의 비트들을 빌려 오면 마스크는 255.255.192.0 (128+64=192)으로 변화된다.

IP주소를 할당할 때 아래와 같은 점을 고려해야 한다.

- TCP/IP는 모든 주콤퓨터들에 통보문을 방송(broadcasting)하기 위하여 매 기계들에 대한 주소를 요구한다. **방송주소**(broadcast address)는 IP주소의 주콤퓨터부를 255로 설정하여 얻는다. 이것은 주소가 192.168.1.0인 망이 방송주소로서는 192.168.1.255, 부분망마스크로서는 255, 255, 255, 0을 가진다는것을 의미한다.
- TCP/IP는 대면부기판이 없는 기계를 **국부주콤퓨터**(localhost)로서 취급하며 이 주콤퓨터는 별도의 주소 즉 **귀환주소**(loopback address)를 요구한다. 이 주소는 127.0.0.1이며 단독주콤퓨터상에서 망봉사를 실행하는데 사용할수 있다.
- 인터네트상에서 사용되는 주소들은 그 망이 인터네트에 접속되지 않았다고 해도 국부망에서 사용되지 말아야 한다. 매 클라스로부터 한 블로크의 주소들이 국부적인 통합망들을 위해 예약되여 있다(표 23-1).

이러한 배경과 함께 망을 위한 주소를 설정할수 있다. 실례로 망주소로서 C클라스주소 192.168.5.0을 선택할수 있다. 그 경우에는 192.168.5.1과 192.168.5.254사이에 주콤퓨터주소를 가질수 있다. 망주소와 방송주소들은 각각 192.168.5.0과 192.168.5.255로 되며 할당에 리용할수 없다.

참고

부분망마스크와 방송주소에서 주콤퓨터부의 대응하는 8비트들의 합은 항상 255로 된다. 그러므로 하나를 알면 다른것을 쉽게 계산할수 있다.

23.2 망대면부기판의 설치

망에 접속해야 할 모든 기계들에는 이써네트기판이 있어야 한다. UNIX체계는 하드디스크와 CD-ROM구동기를 식별하는것처럼 기동시에 망대면부를 식별한다. 대부분의 체계들은 체계설치시에 필요한 구동프로그람과 함께 대면부를 설치한다. 그러나 제작자들도 후에 기판을 추가하기 위한 특수한 도구들을 제공한다.

망대면부기판을 설치할 때 체계는 2개의 하드웨어파라메터를 자동적으로 설정해 볼수도 있으나 그것을 할수 없으면 관리자가 그것들을 제공해야 한다. 일반적으로 UNIX기계가 망에서 정확하게 동작하기 위해서 설정되여야 할 파라메터들은 다음과 같다.

- I/O주소(하드웨어파라메터)
- 새치기벡토르(하드웨어파라메터)
- IP주소
- 부분망마스크

- 방송주소
- 관문주소
- 주콤퓨터이름
- 령역이름

IP주소는 16진수(보통 0x300)이다. IRQ는 옹근수(보통 2 혹은 9)이다. 만일 기계가 이 파라메터들에 자동적으로 값을 설정한다면 별로 걱정할 필요가 없다. 그러나 그렇지 않다면 기판상에 자체로 그것들을 설정해야 한다. PC들에서는 카드가 기계에 삽입된후에 제작자에 의하여 제공된 작은 DOS프로그람을 실행시켜야 할수도 있다. 관리자는 I/O주소와 IRQ가 체계안의 다른 장치(건반, 하드디스크, 직렬포구 등)들이 사용하고 있는것과 충돌하지 않는가를 확인해야 한다.

이 두개의 파라메터가 제공된후에 기계는 기동시에 그 기판을 정확히 식별하여야 한다. 많은 체계들이 기동통보문을 간단히 현시하는 dmesg를 지원한다. 만일 대면부가 제대로 설치되였다면 dmesg출력에서 아래와 같은것을 볼수 있다.

```
ne2k-pci.c:vpre-1.00e 5/27/99 D. Becker/P. Gortmaker
ne2k-pci.c: PCI NE2000 clone 'RealTek RTL-8029' at I/O 0x6200, IRQ 10.
eth0: RealTek RTL-8029 found at 0x6200, IRQ 10, 00:80:c8:02:24:78.
```

여기서 체계는 NE2000 PCI망기판을 인식하며 I/O주소가 0x6200, IRQ가 10으로 설정되였다는것을 보여 준다. 기판의 MAC주소도 마지막에 보여 준다. 이 대면부는 이름을 가지고 있으며 체계에는 eth0으로 알려 진다. 많은 망도구들이 출력에서 이 이름을 특별히 현시한다. 이 기계가 또 다른 대면부를 가지고 있다면 그것은 eth1로 이름 지어 졌을것이다. eth는 Linux가 대면부기판을 이름 짓기 위해 사용하는 접두사이다. 그러므로 이것은 틀림없이 Linux체계이다. 다른 UNIX체계들은 기판이나 소편의 제작자로부터 넘겨 받은 이름형태에 따른다. 실례로 ie0, en0, le0 등과 같은 이름들이 있다.

이제는 구성의 2번째 부분 즉 쏘프트웨어부분으로 가보자. 4개의 8비트들의 묶음 즉 IP주소, 부분망마스크, 방송주소와 관문주소(gateway address)를 정의해야 한다. 기동시에 ifconfig와 route지령이 이 정보를 사용한다. 기계는 또한 관리자가 제공하는 입력으로부터 기계의 FQDN을 설정하기 위해 hostname지령을 사용한다. 부분망마스크와 방송주소는 때때로 망이 부분망으로 되지 않았을 때에는 관리자가 제공한 IP주소로부터 체계가 자동적으로 결정한다.

기계가 관문이나 경로기에 접속되였다면 관문주소를 제공해야 한다. TCP/IP는 망기술에 의하여 전통적으로 만들어 진 관문과 경로기사이의 차이를 없애 버린다. 관문은 때때로 인터네트나 다른 망에 접속하려는 주콤퓨터의 경로일수 있다. 그것은 보통 전용장치이지만 2개의 대면부기판를 가진 기계가 관문으로써도 동작할수 있다. 사실상 한 기계가 4개의 망에 접속되면 그것은 틀림없이 4개의 그러한 기판을 가진다.

23.3 망대면부의 구성(ifconfig)

TCP/IP가 망하드웨어와 독립이므로 IP주소들은 핵심부안에 만들어 지지 않고 망쏘프트웨어안에 존재한다. 대면부의 주소를 설정하기 위해서는 ifconfig지령을 사용해야 한다. 지령은 다음과 같이 사용된다.

```
ifconfig eth0 192.168.0.3
```

이 문법은 지령의 뒤에 대면부이름(여기서는 eth0)과 IP주소가 놓일것을 요구한다. 이 지령은 대면

부에 IP주소를 설정할뿐만아니라 그것을 얻기도 한다. Linux기계가 아니라면 그 기계를 망에서 사용할 수 있게 준비시키는데 이 지령으로 충분하다. ifconfig는 체계파일을 변화시키지 않기때문에 대면부를 영구적으로 설정하지는 않는다. 이 지령은 체계가 기동될 때마다 rc스크립트로부터 실행되여야 한다. ifconfig는 선택적인 인수로써 부분망마스크와 방송주소를 사용한다. 망이 부분망으로 되지 않았으면 부분망마스크와 방송주소가 필요하지 않으므로 여기서는 그것들을 제공하지 않는다. 망이 클라스규칙(A, B, C)을 따르면 ifconfig는 부분망마스크와 방송주소를 자동적으로 계산한다. 부분망으로 된 망에서는 netmask와 broadcast파라메터가 따로따로 제공되여야 한다.부분망안에서 주콤퓨터를 위하여 ifconfig를 어떻게 사용해야 하는가를 아래에 보여 주었다.

```
ifconfig le0 147.35.3.45 netmask 255.255.192.0 broadcast 147.35.63.255
```

여기서 사용하고 있는 대면부는 le0이다. 비록 방송주소가 부분망마스크로부터 유도된다 할지라도 여기서는 방송주소도 지정해 주었다(주콤퓨터부의 대응하는 비트들의 합은 항상 255로 된다). 많은 이전의 BSD 4.2체계들이 방송비트를 설정하기 위하여 1대신에 0을 사용한것만큼 이 방법으로 그것을 보관하는 것이 항상 안전하다.

체계상의 모든 대면부들에 대하여 더 잘 알기 위해서는 사용하고 있는 체계에 의존하여 ifconfig를 사용하거나 -a선택항목을 제공해야 할수도 있다. 아래의 출력은 System V가 가동하는 기계상에서 얻어진것이다.

```
# ifconfig -a
le0: flags=4043<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 147.35.3.45 netmask ffffff00 broadcast 192.168.0.255
      perf. params: recv size: 4096; send size: 8192; full-size frames: 1
      ether 00:20:18:62:47:e0
lo0: flags=4049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
      inet 127.0.0.1 netmask ff000000
      perf. params: recv size: 57344; send size: 57344; full-size frames: 1
```

출력은 체계에 따라 여러가지이지만 이 출력으로부터 대면부의 모든 정적파라메터를 알수 있다. 여기서 ifconfig는 기계의 IP주소뿐아니라 그의 상태도 알려 준다. 체계가 설정되였고(UP) 방송을 지원하며(BROADCAST) 현재 가동중(RUNNING)이다. 그것은 또한 대면부카드의 MAC주소도 보여 주고 있다.

아직 또 하나의 대면부 lo0이 남아 있다. 이것은 모든 주콤퓨터들이 가져야 할 귀환대면부(loopback interface)이다. ifconfig가 IP주소를 127.0.0.1로 명백히 설정하지는 않았지만 그것은 사실상 체계설치시에 수행되였다.

ifconfig는 특정한 대면부이름과 함께 호출될수 있다. Linux는 지어 대면부의 하드웨어파라메터(IRQ와 기초주소)들을 제공한다.

```
# ifconfig eth0

th0      Link encap:Ethernet  HWaddr 00:20:18:62:47:E0
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:9 errors:0 dropped:0 overruns:0
```

```
Interrupt:9 Base address:0x300
```

ifconfig는 단순히 대면부속성들을 설정하거나 현시하는것뿐만아니라 대면부를 활성화 및 비활성화 하는데도 사용될수 있다. 그것은 때때로 IP주소가 변화되여야 할 때 필요하다. 그 지령은 다음과 같은 방법으로 사용된다.

```
ifconfig eth0 down              대면부동작
ifconfig eth0 up              대면부동작
ifconfig eth0 192.168.0.5 up              대면부설정 및 동작
```

ifconfig는 사용자가 같은 망의 모든 주콤퓨터들에 접속할수 있게 하여 주지만 아직까지 다른 망의 주콤퓨터들에는 접근할수 없게 한다. 다른 망의 주콤퓨터들에 접근하려면 체계에 그 망에로의 경로가 제공되여야 한다. 경로조종에 대하여서는 후에 간단히 설명한다.

Linux

ifconfig명령문을 포함하는 rc스크립트(또는 기동시에 실행되는 다른 스크립트)를 보면 그것이 흔히 명백한 값이 아니라 변수들을 인수로 하여 사용되는것을 볼수 있다. 이 스크립트들은 일반적인 의미를 가지며 모든 변수들은 다른 파일에서 정의된다. 변수들은 점(.)이나 source지령(17.9)과 함께 그것들을 포함하는 스크립트를 먼저 실행시키는 방법으로 그 스크립트에 들어 간다. 대부분의 Linux변종들도 이러한 방법으로 진행한다

23.4 망의 검사(ping)

일단 망대면부가 구성되면 그 망에서 작업하고 있다고 알려 진 기계에 파케트들을 보내야 한다. 망의 고장수리에 대체로 사용하는 지령은 ping이다. 이 지령은 원격목적지에 56byte크기의 파케트를 보내며 목적지에서는 그 파케트를 수신하면 응답을 보낸다.

```
# ping 192.168.0.4
PING 192.168.0.4 (192.168.0.4): 56 data bytes
64 bytes from 192.168.0.4: icmp_seq=0 ttl=255 time=1.442 ms
64 bytes from 192.168.0.4: icmp_seq=0 ttl=255 time=0.735 ms
64 bytes from 192.168.0.4: icmp_seq=0 ttl=255 time=0.708 ms
64 bytes from 192.168.0.4: icmp_seq=0 ttl=255 time=0.711 ms
64 bytes from 192.168.0.4: icmp_seq=0 ttl=255 time=0.744 ms
[Ctrl-c]              중단되였다고 표시한다
-- 192.168.0.4 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.708/0.868/1.442 ms
```

매행에서 보여 준 시간은 특수한 파케트(ICMP)가 목적지에 도달했다가 되돌아 오는데 걸리는 왕복시간(round-trip time)이다. 여기서는 응답시간이 빠르고 파케트분실이 없는 잘 접속된 국부망에 대하여 본다. 광지역망에서는 왕복시간이 몇미리초의 범위(수신가능한 령역안에 있는)에 있게 된다. 파케트가 무질서하게 도착했다면 일부 파케트들이 분실될수 있다. 그러나 TCP/IP는 망에서 파케트가 분실되여도 자

료를 전송할수 있게 설계되여 있으므로 파케트분실을 자료분실로 여길 필요는 없다.

주콤퓨터가 정지되였거나 파케트들을 접수할수 없는 경우에는 새치기건을 누른후 다음과 같은것이 현시된다.

```
-- 192.168.0.3 ping statistics ---
50 packets transmitted, 0 packets received, 100% packet loss
```

주해

ping에 의한 검사는 상대측에서 어떤 봉사프로쎄스가 실행될것을 요구하지는 않는다. 그렇지만 ping출력이 성공하였다고 하여 반드시 봉사들이 실행하고 있다는 뜻은 아니다. 실례로 원격기계상에서 inetd가 시동되지 않는다면 ping이 성공이라고 현시되여도 ftp나 telnet은 동작하지 않는다.

참고

ping검사의 실패는 접속과 관계 없다. 원격주콤퓨터가 정지되였거나 망으로부터 림시 차단되였을수 있다. 또 다른 주콤퓨터에서 ping검사를 해보는것이 좋다.

23.5 경로조종

어떤 파케트가 다른 망의 주콤퓨터를 목적지로 하고 있다면 그 파케트는 경로기에 의해 조종되여야 한다. 망의 매 주콤퓨터는 적어도 3개의 마당 즉 대면부의 이름, 목적지의 주소와 경로기의 주소를 포함하는 **경로조종표**(routing table)를 핵심부안에서 관리한다. IP가 파케트를 수신하였을 때 목적지주소의 망부분을 얻어 내기 위해 부분망마스크를 사용한다. 그다음 이 주소를 표에 있는 모든 항목과 비교한다(IP파케트는 이 마스크를 포함하지 않는다).

IP가 경로조종표에서 그 주소를 얻으면 그 파케트를 지정된 관문을 통하여 발송한다. 그러나 발견하지 못하면 파케트를 **기정경로**(default route)를 통하여 보내야 하는데 그것도 표에서 지정된다. 때문에 IP는 단지 주콤퓨터자체에로의 경로보다 오히려 망에로의 경로를 지정한다. 이것이 IP의 가장 중요한 특징의 하나이다.

경로기가 전혀 없다고 해도 모든 망은 최소의 경로조종표를 가지고 있다. 소규모망에서와 같이 주콤퓨터에로의 경로가 제한되여 있을 때에는 체계관리자가 **정적경로조종표**(static routing table)를 만들수 있다.

그렇지만 인터네트에서는 종종 한개의 주콤퓨터에 여러개의 경로가 있게 된다는데로부터 **동적경로조종**(dynamic routing)을 사용한다. 여기서 경로조종데몬(routed와 같은)은 동적경로조종표를 구성하기 위하여 다른 기계상의 경로조종데몬과 통신을 진행한다. 망구성방식이 변화되거나 그 망의 어떤 부분이 정지되면 경로들이 동적으로 조종된다. 경로조종규약은 주콤퓨터에 대한 최소비용경로를 결정할수도 있다. 그것들은 IP가 지능적인것처럼 보이게 한다.

정적경로조종표의 구성(route)

ifconfig지령은 한개 망에 있는 주콤퓨터들이 쓸수 있는 최소경로조종표를 구성한다(Linux는 제외). 다른 망의 주콤퓨터에로 파케트를 보내려면 관문에 대한 경로조종정보가 경로조종표에 추가되여야 한다. 만일 동적경로조종을 포함하지 않는다면 관리자가 수동으로 정적경로조종표를 구성해야 한다. 이것은 route지령으로 수행한다.

정적경로조종표를 어떻게 설치하는가를 보기 위하여 C클라스망 192.168.0.0과 그에 속하는 주콤퓨터 sunny(주소 192.168.0.10)를 고찰하자. 이 망은 2개의 관문 즉 인터네트에 접근하는 michael(192.

168.0.1)과 다른 망 172.16.1.0에 접속되는 fredo(192.168.0.20)를 가진다. 2개의 관문을 통과하는 경로를 설치하기 위하여 sunny의 경로조종표를 구성하여야 한다. michael과 fredo의 망대면부가 이미 ifconfig에 의해 구성되였다고 가정한다.

sunny에서 route지령을 사용하여 먼저 관문 fredo에로의 경로를 설치한다.

```
route add 172.16.1.0 192.168.0.20
```

경로조종표에 경로 172.16.1.0을 추가(add)하고 파케트를 보내기 위하여 접속된 관문이 192.168.0.20(fredo의 주소)이라는것을 지정하였다.

michael은 인터네트에로의 관문으로서 동작한다는데로부터 분명히 더 많은 경로를 조작하며 따라서 기본관문으로 설정되여야 할것이다. 이것은 경로지정대신에 예약어 default를 써서 수행한다.

```
route add default 192.168.0.1
```

이렇게 일단 기본경로를 설정하면 국부망과 172.16.1.0으로 향하지 않는 모든 파케트들은 이 관문(192.168.0.1)에로 발송된다. 그때 그 망에 지장없이 접근할수 있다. 물론 이것은 그 관문기계가 IP발송을 지원하는 방화벽으로서 또는 대리봉사기(이 책에서는 고찰하지 않는다.)로서 구성되며 주콤퓨터가 이 기계에 파케트를 발송하도록 허락하였다고 가정한다.

경로를 삭제하기 위해서는 예약어 delete를 사용한다. 우에서 생성된 경로들은 예약어 add를 간단히 delete로 바꾸어 삭제될수 있다.

```
route delete 172.16.1.0 192.168.0.20
route delete default 192.168.0.1
```

Linux에서 route지령은 좀 다르다. 여기서는 관문을 가리키는 예약어 gw를 사용한다. Linux에서 route add와 route delete는 다음과 같이 입력되여야 한다.

```
route delete 172.16.1.0 gw 192.168.0.20          Linux는 gw를 사용한다
route add default gw 192.168.0.1
```

route는 망이 부분망으로 되지 않는 한 부분망마스크를 알 필요가 없다. 그러나 부분망으로 되였을 때는 netmask예약어를 써서 부분망마스크를 지정하여야 한다. 때때로 망경로를 참조하고 있다는것을 route에 알리기 위하여 목적지주소와 함께 예약어 net를 사용해야 한다. 이따금 주콤퓨터경로를 지정하도록 요구할수도 있으며 그 경우에는 host예약어를 사용해야 한다. Linux와 Solaris는 그대신 -net와 -host를 사용한다.

ifconfig로 하여 이 route명령문들은 기계가 가동하고 있는한 유효하다. 그것들은 체계가 기동될 때마다 실행되여야 하며 따라서 반드시 체계의 기동스크립트들속에 보존된다.

23.6 망파라메터의 현시(netstat)

netstat -rn지령은 경로조종표를 현시한다. -r선택항목은 경로조종표를 렬거하며 -n은 IP주소들의 수값형식을 현시한다. 앞의 절에서 본 2개의 route add문을 Linux체계에서 실행하면 다음과 같이 된다.

```
# netstat -rn
kernel IP routing table
```

```
Destination   Gateway        Genmask          Flags  MSS Window  irtt Iface
172.16.1.0    192.168.0.20   255.255.255.255   UGH    0  0          0 eth0
192.168.0.10  0.0.0.0            255.255.255.255   UH     0  0          0 eth0
192.168.0.0   0.0.0.0        255.255.255.0     U      0  0          0 eth0
127.0.0.0     0.0.0.0        255.0.0.0         U      0  0          0 lo
0.0.0.0       192.168.0.1    0.0.0.0           UG     0  0          0 eth0
```

첫행은 망 172.16.1.0을 목적지로 하는 임의의 파케트가 IP주소 192.168.0.20을 가진 기계에로 발송된다는것을 보여 준다. Flags아래에 있는 G는 관문에 련결된다는것을 가리키며 그것이 없으면 주콤퓨터들에 직접 련결된다는것을 가리킨다. 여기서 망주소를 제공하면 경로조종표의 크기를 줄일수 있다.

3번째 행은 망 192.168.0.0으로 향한 파케트들이 기계 0.0.0.0(그 기계의 경로조종표가 현시되였다.)에로 발송된다는것을 보여 준다. 이 기계는 자체의 대면부를 리용하여 같은 망의 다른 기계들과 통신한다. 파케트의 목적주소가 첫렬에 있는 항목들중 어느것과도 맞지 않는다면 그때는 마지막행으로 결정된다. 여기서 목적지가 0.0.0.0이므로 기정관문 192.168.0.1이 사용된다. 여기서 0.0.0.0은 이전의 어느 주소와도 대응되지 않는 주소들을 표현한다(case에서 사용되는 *와 같다).

망의 모든 주콤퓨터들은 2개의 대면부 즉 이써네트주소(etho)와 귀환주소(lo0)를 가지고 있다. 모든 대면부들이 설정되였고 가동중에 있다(기발은 U를 보여 준다). H는 그 경로를 통하여 오직 하나의 주콤퓨터에만 갈수 있다는것을 가리킨다. 국부주콤퓨터(127.0.0.0)에로 향한 파케트는 관문으로서 같은 기계(0.0.0.0)를 사용한다. 결과적으로 이 지령이 실행된 기계는 IP주소 192.168.0.10을 가지고 있다.

주해

기계가 어떤 주콤퓨터에도 접속되지 않았다면 경로조종표는 귀환대면부를 위한 항목만을 보여 준다. 경로조종표에 이 경로가 있으면 그 주콤퓨터상에서 TCP/IP 도구들을 사용할수 있다.

23.7 인터네트데몬(inetd)

UNIX는 매 의뢰기로부터 오는 요청을 위해 각각의 지정된 포구번호를 감시하는 여러개의 데몬들을 가지고 있다. 이 모든 데몬들을 늘 실행시켜야 하겠는가(비록 일부 데몬들이 대부분의 시간동안에는 리용되지 않는다 할지라도)? 꼭 그런것은 아니다. 요청이 제기되였을 때 그 요청만큼 해당한 데몬들을 불러내는것이 적당하다. 이러한 경우에 inetd데몬을 사용한다.

telnet와 ftp봉사를 위한 데몬들을 비롯하여 많은 TCP/IP데몬들은 의뢰기에 의해서 시동되는것도 아니고 rcn.d안에 있는 체계의 시동스크립트에 의해 시동되는것도 아니라 기본인터네트데몬(master Internet daemon)인 inetd에 의하여 시동된다. inetd는 임의의 접속요구들을 위하여 여러개의 포구들을 감시한다. 요구를 검출하면 구성파일 /etc/inetd.conf안에서 그 포구에 대하여 정의된 프로그람을 시동한다. 이 파일에는 한 행당 한 봉사가 들어 있다.

```
ftp     stream  tcp  nowait  root    /usr/sbin/tcpd  in.ftpd -l -a
telnet   stream  tcp  nowait  root    /usr/sbin/tcpd  in.telnetd
talk    dgram  udp  wait    nobody  /usr/sbin/tcpd  in.talkd
pop-3   stream  tcp  nowait  root    /usr/sbin/tcpd  ipop3d
#imap  stream  tcp  nowait  root    /usr/sbin/tcpd  imapd
```

```
#tftp   dgram  udp  wait    root    /usr/sbin/tcpd  in.tftpd
```

첫렬은 봉사를 보여 준다. 그 봉사가 사용하는 통신규약은 3번째 렬에서 보여 준다. 4번째 렬이 nowait이면 그것은 같은 봉사를 위하여 여러개의 접속이 이루어 질수 있다는것을 의미한다. 마지막 2개의 렬은 봉사기프로그람의 절대경로이름과 봉사기가 최종적으로 불러 내는 프로그람의 완전한 지령행을 보여 준다.

이 모든 행들은 공통점을 가진다. 그것들은 inetd에 의해서 직접 호출되지 않고 tcpd라고 불리우는 외피프로그람(wrapper program)을 통하여 호출된다. tcpd는 먼저 의뢰기가 이 봉사를 사용할 권한이 있는가를 구성파일에서 검사하고 그다음 대응하는 봉사기프로그람을 불러 낸다. ftp에서는 이 데몬이 봉사기프로그람 in.ftpd이며 그것은 인수 -l -a와 함께 실행된다. tcpd는 또한 별도의 파일에 그 요구를 기록한다. tcpd를 통한 망접근은 파일 hosts.deny와 hosts.allow에 의하여 조종된다.

UNIX체계는 여러개의 봉사를 시작하지만 그중 많은것들이 전혀 요구되지 않을수도 있다. 그 경우에 그 행을 설명문으로 만들어 봉사 그자체를 금지시키는것이 더 좋다. 이제는 왜 거의 모든 UNIX체계상에서 tftp(trivial file transfer protocol)가 초기에 실행되지 않는가를 리해할수 있을것이다.

ftp가 어느 포구를 사용하는가 하는것은 /etc/services를 탐색하여 결정한다. 이 파일은 첫번째 마당 즉 봉사이름을 통하여 inetd.conf와 련관된다. 그 파일에는 2개의 마당이 들어 있다.

```
tfp      21/tcp
telnet    23/tcp
smtp    25/tcp      mail
pop3    110/tcp      # POP version 3
pop3    110/udp
```

포구번호는 통신규약끄리표를 가지고 있다. 이 표를 탐색하면 봉사에 의해 사용되는 포구번호뿐아니라 통신규약도 결정할수 있다. 이 파일안에 있는 많은 봉사들이 2개의 항목 즉 tcp에 관한것과 udp에 관한것을 가진다.

주해

tcpd프로그람은 모든 체계들에서 다 쓰는것은 아니다. 일부체계(Solaris)들에서 봉사기프로그람은 inetd에 의하여 직접 시동된다. ftp봉사에서 이것은 inetd가 외피프로그람 tcpd를 통하지 않고 직접 in.ftpd(이전의 체계들에서는 ftpd)를 시동한다는것을 의미한다. 그 경우에 /etc/inetd.conf의 마지막 2개렬은 같은 지령이름을 보여 준다.

23.8 점대점규약(pppd)

사용자의 콤퓨터가 망기판을 가지고 있지 않을수도 있고 또는 사용자가 외부세계와의 통신수단으로 전화만을 가지고 있는 곳에서 생활할수도 있다. 사용자는 2대의 기계를 직렬포구(DOS에서는 COM1과 COM2)들을 통하여 련결하고 이 련결상에서 TCP/IP를 실행시킬수 있다. 이때 특수한 통신규약 즉 **점대점규약**(Point-to-Point Protocol: **PPP**)을 리용한다. PPP는 직렬포구통신공간에서 SLIP와 UUCP를 대신한다. 오늘날 PPP는 사용자들이 자기 콤퓨터의 직렬포구상에 설치된 모뎀을 사용하여 인터네트에 접근하는 표준방식이다(더 새로운 기술에 기초한 방법들도 만들어 지고 있지만).

PPP는 류다른 통신규약이다. 이것은 IP주소묶음을 리용하여 두개의 주콤퓨터사이에 접속을 설정하므로 ftp나 telnet와 같은 다른 통신규약들이 련이어 실행될수 있다. 통신프로쎄스에서 PPP는 련결의 지

속성을 확인하는것밖에 다른 부분은 없다. 또한 의뢰기와 봉사기구성요소를 따로따로 가지고 있지 않다. 하나의 pppd지령을 의뢰기로서 또는 봉사기로서 동작하도록 적절한 선택항목들과 함께 호출해야 한다. 이 절에서는 Linux나 많은 UNIX체계(Solaris는 제외)들에서 사용되는 BSD PPP묶음을 설명한다.

Linux체계에서 pppd지령은 /usr/sbin등록부안에 들어 있다. 일반사용자등록자리로 pppd를 실행하자면 몇가지 구성이 요구되므로 여기서는 뿌리사용자가 가입한다고 가정한다. 사용자가 자기의 ISP에 접속하고 있다면 국부기계상에서 아래와 같이 pppd프로쎄스를 시동해야 한다.

```
/usr/sbin/pppd /dev/ttyS0 115200 crtscts modem defaultroute noipdefault -detach
```

/dev/ttyS0은 첫번째 직렬포구(COM1)에 접속한 모뎀장치이며 포구속도는 모뎀이 조작할수 있는 최대속도의 3배로 설정된다(여기서는 115,200보 즉 초당 115,200bit). 그 리유는 뒤에서 설명한다. 이 지령은 또 다른 대면부(ppp0)를 설정하는데 그의 속성은 ifconfig와 netstat -rn지령으로 현시할수 있다.

인터네트에 접근하기 위해서는 사용자의 콤퓨터(의뢰기)와 ISP(봉사기)사이에 PPP를 실행시켜야 한다. 이것은 류사한 PPP프로쎄스가 상대측에서도 실행되여야 한다는것을 의미한다. 의뢰기의 PPP대면부가 별개의 IP주소가 없이 설정되면(우에서와 같이) 봉사기는 련결을 수립하기전에 IP주소를 제공해야 한다.

참고

관리자가 항상 고정된 한 묶음의 선택항목들로 pppd를 사용하여야 한다면 pppd의 구성파일 /etc/ppp/options안에 이 선택항목들을 거의 모두 보관할수도 있다. 이 파일에는 행당 하나의 선택항목이 들어 있다.

pppd는 복잡한 지령행을 가지며 chat와 같은 도구를 리용하여 수동적으로 프로쎄스를 시동해야 하는 경우에는 이 복잡한 문법을 알아 두는것이 더 좋다. 또한 왜 이 선택항목들을 가지고 우의 지령을 호출해야 하는가도 리해하여야 한다.

- PPP는 모든 8비트들을 사용하므로 **하드웨어흐름조종**(hardware flow control)이 사용되여야(crtscts) 체계가 자료의 흐름을 조정할수 있다.
- **쏘프트웨어흐름조종**(software flow control)은 차단되여야 한다. 이때 모뎀이 문자 [Ctrl-s]와 [Ctrl-q]을 시작문자와 끝문자로 해석하지 않도록 하여야 한다. 그렇게 하자면 pppd가 xonoff 선택항목을 사용하지 말아야 한다.
- 국부망에로 향하지 않는 IP파케트를 위한 기정경로(default route)는 PPP대면부를 통하여 구성되여야 한다.
- 대부분의 ISP들은 **동적주소화**(dynamic addressing)기능을 가지는데 그것은 봉사기가 의뢰기의 IP주소를 제공해야 한다는것을 의미한다(noipdefault). 또한 ppp는 IP주소를 자체로 설정하는것도 허용한다.
- pppd가 말단으로부터 분리되는것이 허용되지 말아야 하며 그렇지 않으면 접속이 지속되지 않을것이다(-detach).

여기서는 봉사기측선택항목에 대해서 고찰하지 않는다. 그렇지만 집과 사무실에 있는 기계들사이에서 PPP접속을 설정한다면 그때는 봉사기측의 선택항목도 알아야 한다.

주해

PPP는 대체로 의뢰기측과 봉사기측구성요소를 따로따로 가지지 않는 쓸모 있는 TCP/IP도구이다. 그것은 체계의 시동스크립트에 의해 시동되지도 않으며 inetd에 의해 조종되지도 않는다.

참고

만일 일반사용자등록자리로부터 pppd를 사용하는 경우에는 그의 SUID비트를 설정할수도 있다(22.4). 이것은 chmod a+s /usr/sbin/pppd를 사용하여 수행된다. Linux의 일부 판본들은 사용자들이 pppd를 소유한 집단에 속한다고 생각한다.

23.9 PPP에 의한 인터네트접속

Linux기계를 사용하여 ISP에로 즉 관문을 통하여 인터네트에 접속하여 보자. 사용자는 직렬포구(말하자면 COM1)에 모뎀을 접속하고 그것을 사용하여 전화선으로 ISP에 련결해야 한다. Linux에서 모뎀설치는 간단하며 /dev/modem과 /dev/ttyS0사이의 기호련결을 설정한다(관리자가 첫 직렬포구를 사용하고 있다고 가정한다). /dev/modem의 목록은 다음과 같다.

```
lrwxrwxrwx    1 root   root         10 DEC  5 08:44 /dev/modem -> /dev/ttyS0
```

ISP들은 보통 UNIX기계를 사용하며 사용자는 거기에 등록되여 있는 사용자이름과 통과암호를 리용하여 그 콤퓨터에 가입하여야 한다. 일단 가입하면 봉사기측에서 PPP프로쎄스가 시작되여야 한다. 만일 ISP기계가 그것을 자동적으로 수행하지 않는다면(일반적으로 수행한다.) 그때는 사용자가 그것을 시동해야 한다. 그다음에는 모뎀을 재설정함이 없이 ISP의 기계에서 탈퇴하여 자기의 기계상에서 PPP를 시동해야 한다.

ISP의 PPP프로쎄스는 일반적으로 관리자의 IP주소를 할당한다. 관리자의 기계상에서 pppd를 시작한후 사용자의 콤퓨터와 그 ISP의 주콤퓨터사이에 련결이 수립된다. 이 련결은 또한 인터네트에로의 기정경로(pppd에서 defaultroute선택항목)를 제공한다. 이제는 사용자의 콤퓨터가 인터네트의 부분으로 된다. ISP기계는 PPP련결을 제공하는 역할만을 수행한다.

관리자의 ISP는 대체로 하나이상의 **이름봉사기**(name server)도 관리한다. 이름봉사기는 주콤퓨터이름을 IP주소로 분석하는 기계이다. 모든 령역이름들이 분석되여야 하며 그 IP주소들이 관리자의 기계에서 발송될 PPP파케트내부에 삽입되여야 하므로 이름봉사기에로의 접근이 필요하다. ISP는 또한 전자우편봉사기를 제공하며 사용자가 새소식그룹에 접근할수 있도록 새소식봉사기를 관리한다.

23.9.1 이름봉사기와 분석기의 지정

사용자의 ISP는 이름봉사기들의 IP주소를 제공하여야 한다. 관리자는 Linux체계상에서 2개의 파일들을 수정해야 한다. /etc/host.conf안에 아래의 2개 행을 간단히 삽입하여야 한다.

```
order hosts bind              이름봉사기는 BIND로 동작
multi on
```

분석기(rosolver)는 응용프로그람을 대신하여 FQDN을 IP주소로 변환하여 줄것을 이름봉사기에 요구하는 의뢰기이다. UNIX체계상에서 분석기는 구성파일 /etc/resolv.conf를 사용한다. 이 파일에 사용자의 ISP의 이름봉사기주소를 배치하여야 한다.

```
nameserver 202.54.1.30
nameserver 202.54.9.1              첫번째와 두번째 이름봉사기의 IP주소
```

이것이면 분석기구성이 완성된다. 그다음 기계는 FQDN을 분석하기 위하여 BIND(/etc/hosts가 실패한 후에)를 사용해야 한다는것을 알게 되며 어느 기계들이 이 목적을 위해 접속하려는가도 알게 된다.

23.9.2 스크립트파라메터얻기

인터네트에 접속하기 위해서는 2개의 문자기반의 도구들을 사용한다. 사용자는 스크립트를 통하여 원격말단에로 모든 입력자료를 보내며 다시 그쪽으로부터 어떤 문자렬들이 올수 있다. 전화회선에 의해 련결한 다음 프롬프트가 나오면 해당한 문자렬들을 입력한다. minicom은 Hayes의 AT지령을 사용할수 있으므로 여기에 맞는 전형적인 도구이다. 이 지령들은 AT로 시작하는 문자렬들이며 모뎀이 번호를 호출하고 초기화도 할수 있다. 련결하기전에 모뎀속도를 조작할수 있는 가장 높은 처리속도로 설정하였는가를 확인하여야 한다. 대부분의 모뎀들과 ISP들은 자료압축을 지원하며 이것은 모뎀의 속도를 향상시킨다(보통 인자 4를 쓴다). 이것은 33.6Kbps모뎀에서 115,200bps의 전송속도를 안전하게 사용할수 있다는 것을 의미한다. 사용자가 minicom과 33.6 또는 56kbps모뎀을 사용할 때 minicom -s를 불러 내면 직렬포구속도를 훨씬 높게 설정할수 있다. minicom에서 탈퇴하고 그다음에 다시 가입한다.

관리자는 한조의 AT지령을 사용하여야 한다. 먼저 atz로 모뎀을 재설정한 다음 atdt와 그뒤에 ISP의 전화번호를 주어 련결한다. 사용자확인을 요구하는 2개의 프롬프트(사용자이름과 통과암호)에 대답을 준다. ISP가 pppd프로쎄스를 자동적으로 시동하지 않는다고 가정하고 자체로 그것을 해보도록 한다(그림 23-1).

```
Press ALT-Z for help on special keys

AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0
OK
atz                                                          모뎀의 초기화
OK
atdt5599001                                           전화번호는 5599001
CONNECT 28800/ARQ/V34/LAPM/V42BIS
User Access Verification

Username: sumit                                          첫번째 입력문자렬
Password: *********                                      두번째 입력문자렬
gicaro31> ppp                                            세번째 입력문자렬
Entering PPP mode.        봉사기 PPP는 ppp로 수동적으로 기동된다는데 주의를 돌리시오
Async interface address is unnumbered (Ethernet0)
Your IP address is 202.54.52.240. MTU is 1500 bytes
~~y}#.!}!q} }4}"}&} }*} } }%}&Tb..}'}"}()"!. ~~y}#.!}!r} }4}"}&} }*} } }%}&Tb..}~
~~y}#.!}!q} }4}"}&} }*} } }%}&Tb..}'}"}()"!. ~~y}#.!}!r} }4}"}&} }*} } }%}&Tb..}~
.....
```

그림 23-1. ISP의 PPP봉사기에 가입하기

그림은 동적IP주소달기를 작업화면으로 보여 준다. ISP는 관리자에게 주소 202.54.52.240을 할당하였다. 그림의 아래부분에 보이는것은 봉사기측에서 발생한 간단한 PPP파케트들이다. 이 출력시 관리자는 3개의 문자렬 Username:, Password:, gicaro31>이 온다는데 주의를 돌려야 한다. 관리자가 돌려 보내야 할 문자렬들이 그옆에 있다. 이러한 정보를 리용하여 스크립트를 개발하여 보자.

23.9.3 dip의 사용

문자기반의 회선접속도구들속에서 dip는 SunOS, AIX, Ultrix, Linux와 같은 넓은 범위의 UNIX체

계들속에서 쓸모가 있다. dip는 프로그람작성구조와 거의 비슷하며 론리적인 오유조종을 할수 있다. dip는 프로그람화되므로 재접속할수도 있다. 송신문자렬과 dip를 사용한 스크립트를 다시 보자.

```
# cat dipdial.dip              일반적으로 dip확장자를 사용
get $local 0.0.0.0             국부IP주소는 동적이다
port modem                     기호련결이 설치되여야 한다
speed 115200                   33.6kbps모뎀으로 권고
dialstart:
reset
flush
send atdt5599001\r             번호 5599001을 사용할수도 있다
sleep 2
wait CONNECT
wait name: 20                  name: 프롬프트에 대하여 20초까지 대기
if $errlvl != 0 goto redial    모든것이 제대로 되여 있지 않다면
send sumit\r
wait word:
send a9h4uil\r                 암호는 볼수 있다
wait >
send ppp\r                     봉사기에서 pppd를 시동한다
mode PPP                       의뢰기에서 pppd를 시동한다
exit
redial:
sleep 1
goto dialstart
```

이 스크립트는 거의 모두가 이미 나온것이므로 간단히 설명한다. 여기서 [Enter]건은 echo에서도 리해되는 확장문자렬 \r로 표시된다. 프롬프트문자렬안에서 미세한 변화를 허용하기 위하여 부분문자렬들을 사용한다. 실례로 name:은 username:과 Username: 둘 다 허용한다. $errlvl변수는 련결이 성공하였을 때 0으로 설정된다. if문은 오유가 일어 나면 모뎀이 다시 번호를 호출하도록 해준다. dip는 의뢰기측에서 간단한 명령(mode ppp)을 가지고 pppd프로쎄스를 시동한다. 이것은 앞에서 보여 준 pppd지령행을 시동하는것과 같다.

관리자가 이것을 파일 dipdial.dip에 보존하여 놓은 다음에는 dip지령을 실행하여야 한다.

```
/usr/sbin/dip -v dipdial.dip
```

이것으로 조작이 끝난다. 이것은 모뎀접속을 진행하고 관리자를 가입시키며 량측에서 PPP프로쎄스들을 시동하고 관리자에게 프롬프트를 돌려 준다. 이제는 인터네트상에서 임의의 인터네트도구들과 이 장에서 취급한것들을 사용할수 있을것이다. pppd를 끝내려면 dip -k를 사용한다. 이 지령이 동작하면 관리자의 통과암호를 화면상에 현시하기때문에 -v(verbose)선택항목을 삭제하여야 한다.

주해

만일 봉사기가 자체로 PPP를 시동하면 두개의 명령문 즉 wait >와 ppp\r을 이 스크립트에서 삭제해야 한다.

참고

만일 사용자가 임풀스식전화체계를 가지고 있다면 atdt대신에 atdp를 사용해야 한다. 더 느린 모뎀을 사용한다면(말하자면 14.4kbps정도의) 속도를 38,400으로 설정할수 있다. 모뎀이 접속되지 않는다면 atdt대신에 atx3dt를 사용하시오. 이것이 실패하면 Linux체계상에서 가능한껏 wvdial과 wvdialconf지령을 사용하시오.

23.9.4 chat의 사용

체계가 dip를 가지고 있지 않다면 관리자는 chat지령을 쓸수 있다. chat도 스크립트를 사용하며 chat스크립트안의 매행(ABORT를 포함하는 몇개 행은 제외)에는 송신문자렬조가 들어 있다. chat는 훨씬 단순하고 덜 정교한 스크립트를 사용한다.

```
# cat chatdial.chat
'' atz                      atz를 사용하기전에 요구하는것이 없다
OK atdt5599001              요구한다면 접속한다
ABORT BUSY                  모뎀이 BUSY인가 혹은
ABORT 'RING - NO ANSWER'    RING-NO ANSWER인가 검사한다
CONNECT ''                  CONNECT를 요구하나 수신이 없다
name: sumit                 3개의 수신문자렬조
word: a9h4uil
> ppp                       봉사기가 자체로 pppd를 시동하므로 필요 없다
```

빈 문자렬(null string)이나 빈 응답은 ''로 표시한다. 이 스크립트는 모뎀이 내보낼수 있는 BUSY와 RING - NO ANSWER통보문을 관리하며 예약어 ABORT를 사용하여 완료한다. 이 스크립트는 의뢰기측에서 PPP를 시동하지 않는다. 여기서는 서로 다른 접근방법을 받아 들인다. 즉 pppd지령을 실행시키고 그의 connect선택항목을 사용하여 chat스크립트를 실행시킨다. 모든 조작은 배경에서 진행할수 있다.

```
/usr/sbin/pppd /dev/ttyS0 115200 connect "/usr/sbin/chat -f chatdial.chat" \
crtscts modem defaultroute noipdefault -detach  &
```

dip와 달리 chat에는 pppd프로쎄스를 제거하기 위한 선택항목이 없다. ps를 실행시켜 일반적인 방법으로 pppd를 제거할수 있다. 그러나 pppd프로쎄스의 PID가 본문파일 /var/run/ppp0.pid에 저장되므로 다음과 같이 그것을 제거할수 있다.

```
kill -9 `cat /var/run/ppp0.pid`
```

여기서는 재접속을 할수 없으므로 단번에 ISP에 접속하려면 chat를 사용할수 있다. 더 쉽게 조작하기 위해서는 이 절차를 쉘스크립트안에 넣거나 별명으로 구성할수도 있다.

23.9.5 련결을 수립한후에

dip와 chat가 지장없이 통과될수 있지만 아직도 원격싸이트에 접속할수는 없다. 그 경우에 먼저 netstat -rn이나 route -n을 사용하여 사용자의 PPP대면부가 가동중인가를 검사하여야 한다.

```
# netstat -rn               route -n을 사용할수도 있다
kernel IP routing table
Destination   Gateway    Genmask          Flags  Metric Ref  Use Iface
192.168.0.3   0.0.0.0     255.255.255.255 UH     0      0      0 dummy0
```

```
202.54.1.30   0.0.0.0       255.255.255.255 UH    0      0      0 ppp0
192.168.0.0   0.0.0.0       255.255.255.0   U     0      0      0 eth0
127.0.0.0     0.0.0.0       255.0.0.0       U     0      0      1 lo
0.0.0.0       202.54.1.30   0.0.0.0         UG    0      0      0 ppp0
```

ppp0에 속하는 행은 두개이다. 2번째것은 pppd지령과 함께 defaultroute선택항목을 리용하는 방법을 보여 준다. 봉사기를 ping으로 검사하고 이것이 동작하면 그때는 알고 있는 주콤퓨터(말하자면 ping rs. internic.net와 같은)를 ping으로 검사하여 본다. 만일 ping이 일반적인 출력을 내지 못한다면 번호호출스크립트안에 있는 모뎀의 속도설정을 검사해 보아야 한다.

pppd는 련결수립후에 스크립트 /etc/ppp/ip-up을 불러 내며 련결을 끊기전에는 /etc/ppp/ip-down을 호출한다. Web사용에서 활동이 제한된 사용자들은 일반적으로 이 파일을 리용할 필요가 없을수도 있다. 그러나 ip-up은 인터네트상에서 봉사기로부터 우편을 보내고 받는 처리를 자동화하는데 유용하다. 관리자가 자기의 회선접속등록자리를 가지고 있다면 그때는 수동으로 하는것보다 오히려 이 스크립트로부터 sendmail과 fetchmail을 실행하여 접속시간을 절약할수 있다.

참고

관리자는 때때로 모뎀이 잠그어 져 있다(lock)는 통보를 받을수도 있다. 그러한 경우에 오유통보문은 관리자에게 파일 /var/lock/LCK..modem을 삭제할것을 요구한다. 이 지시대로 해보시오. 만일 /dev/modem상의 쓰기비트가 불가능하게 되면 그때는 chmod a+w /dev/modem을 써서 그것을 가능하게 하여야 한다.

23.10 PAP와 CHAP인증

대부분의 ISP들은 주콤퓨터의 신원을 확증하기 위하여 몇가지 형식의 인증을 사용한다. PPP는 두가지 형식의 인증을 지원한다.

- **암호인증규약**(Password Authentication Protocol:**PAP**): 이 체계에서는 인증을 위하여 통과암호본문과 사용자이름이 망을 통하여 전송된다.
- **도전맞잡기인증규약**(Challenge Handshake Autheutication Protocol:**CHAP**): 이것은 망으로 통과암호를 보내지 않는 두방향인증체계이다.

PAP와 CHAP는 봉사기와 의뢰기 둘 다가 아는 **공유비밀**(shared secret)을 사용한다. 이것은 대체로 사용자의 통과암호이며 PAP에서는 파일 /etc/ppp/pap-secrets안에 보관된다. 이 파일안에 들어 있는 표준행을 보기로 하자.

```
henry       starisp.com      my:pass,word
```

첫 마당은 사용자의 주콤퓨터이름이다. 만일 의뢰기가 인터네트상에 없다면(전화선련결의 경우처럼) ISP는 이 이름을 알 필요가 없다. PPP는 주콤퓨터이름이 아니라 ISP에 등록된 사용자이름으로 이 마당을 해석하는데 호출될수 있다. 2번째 마당은 ISP의 FQDN이며 여기서는 하나의 별표(*)가 모든 ISP들에 대응한다. 3번째 마당은 공유비밀이며 이 경우에는 통과암호가 들어 있다. 4번째 마당은 보통 비여 있다.

사용자가 ISP에 련결할 때 통과암호는 명백한 본문형태로 ISP측에 보내여 진다. 이 통과암호가 ISP의 파일 /etc/ppp/pap-secrets안에 들어 있는것과 맞을 때에만 련결이 설치된다.

PAP보안은 보안이 전혀 없는것보다는 낫지만 파케트탐지기(packet sniffer)가 전송되는 본문통과암호를 가로 챌수 있다. 명백히 말하여 전화선상에서보다 이써네트망에서 통과암호를 가로 채는것이 더 쉬

우며 그러한 공격에는 약하다. PAP가 지금도 광범하게 쓰이고는 있지만 CHAP가 보다 안전한 형식의 인증을 제공한다.

PPP는 기정인증규약으로서 CHAP를 사용한다. CHAP는 통과암호를 실제적으로 교환하지 않는다. 오히려 봉사기가 의뢰기에 **도전문자렬**(challenge string)을 보낸다. 의뢰기는 그 도전문자렬과 /etc/ppp/chap-secrets안에 있는것과 류사한 형식으로 들어 있는 공유비밀을 뒤섞어(부호화하여) 응답한다. 봉사기 역시 그 공유비밀을 알고 있으므로 류사한 동작을 진행한다. 그다음 봉사기는 자기가 계산한것과 의뢰기로부터 받은것을 비교한다. 두 형태가 맞을 때 련결된다.

어느 형식이든 인증을 사용하려면 봉사기측의 pppd가 auth(authentication)선택항목과 함께 호출되여야 한다. 그렇지만 의뢰기측은 user선택항목(여기서는user henry)을 가지고 그것을 불러 내야 한다. 인증은 가입에 의해서가 아니라 계산에 의해서 수행되므로 이전의 dip와 chat스크립트들을 수정하여 송신문자렬과 관계되는 모든 코드를 삭제하여야 한다. dip에서 사용자는 CONNECT다음의 모든것을 삭제할수 있으나 mode PPP지령은 남겨 둘수 있다. 더 많은 선택항목을 배치하기 위하여서는/etc/ppp/options파일을 사용하여야 한다. chat스크립트는 CONNECT다음에 아무것도 없어야 한다.

주의

파일 pap-secrets와 chap-secrets는 통과암호나 도전문자렬을 포함하므로 뿌리를 제외한 모든 사용자들이 해석할수 없어야 한다. 그렇지 않으면 체계는 아무런 인증도 없이 사용되기때문에 안정하지 못한다.

23.11 망파일체계

TCP/IP를 써서 원격으로 가입하여 파일을 전송하는 기능이 때로는 충분하지 못하다. 실례로 거대한 자료기지를 전송하는데는 실용적이지 못하다. 만일 원격파일체계가 국부파일체계상에 설치되여 있어서 사용자가 그 원격체계에 접근하는데 특별한 지령을 쓸 필요가 없다면 그것이 훨씬 더 좋을것이다. 이러한 요구로부터 **망파일체계**(NFS:Network File System)라는 개념이 나왔으며 그때로부터 이것은 현실적으로 모든 UNIX변종들에 도입되였다.

망파일체계는 국부등록부에 원격파일체계를 설치한다. 이것은 그것들이 국부적으로 접속된것과 같은 느낌을 준다. 실례로 주콤퓨터 fredo상의 /datab파일체계가 등록부 /oracle상에 국부적으로 설치된다면 사용자는 /oracle이 국부파일체계인지 원격파일체계인지를 모른다. 원격파일체계상의 파일들을 접근하는데는 특별한 지령이 필요 없다.

NFS기능은 여러가지로 쓸모가 있다. NFS는 여러대의 콤퓨터상에서 작업하고 있는 사용자들이 파일들을 공유하도록 한다(이전에는 모든 사용자들이 제가끔 자기의 콤퓨터상에 파일의 복사본들을 만들어야 했다). 이것은 사용자가 다른 사용자들에 의하여 접근될수 있는 몇대의 콤퓨터상에 있는 디스크들을 모두 가질수 있다는것을 의미한다. 또한 체계유지와 여벌복사가 더 쉬워 진다. 왜냐하면 관리자가 여러대의 기계상에 동일한 여벌복사를 하지 않고 파일의 한 묶음만을 복사하여 관리하면 되기때문이다.

국부설치와 달리 NFS는 파일체계만을 설치하도록 제한되지 않는다. NFS는 개별적인 파일체계를 구성하지 않는 임의의 등록부도 어떤 원격등록부에 설치할수 있다. 사용자들은 등록부상에서 읽기 또는 읽기쓰기허가를 받을수도 있다. 사용자는 또한 그 기능을 사용하도록 허용할 주콤퓨터들을 지정할수 있다. 관리자도 이 기능을 사용하기 위하여 주콤퓨터허가를 지정할수 있다. 그러나 NFS는 사용자준위에서 허가권을 주지 않는다. 관리자가 어떤 주콤퓨터에 접근을 허용하였다면 관리자는 그 주콤퓨터상의 모든 사용자들과 협동하게 된다. 이때 NFS가 좀 불안해 진다(특히 쓰기접근을 제공할 때)

주해

관리자가 과부하체계상에서 디스크공간을 초과하여 실행한다면 개별적인 하드디스크를 추가하지 말고 별도의 기계를 추가하고 그 기계의 파일체계를 원격으로 설치할수 있다. 그 우점은 관리자의 체계가 시종일관 가동한다는것이다.

NFS의 설치

NFS는 봉사기측에서 주로 2개의 데몬 mountd와 nfsd(Linux에서는 rpc.mountd와 rpc.nfsd)에 의해 조종된다. mountd는 사용자요구를 확인하며 nfsd는 사실상 파일체계를 태우거나 내리우는것으로써 의뢰기에 봉사를 제공한다.

파일체계를 조종하기 위해 사용되는 지령들은 같지만(mount 와 umount) 여기서는 일부 다른것이 있다. 대부분의 체계들은 파일체계들을 공유하는 Sun의 Solaris가 아니라 파일체계들을 반출하고 BSD형의 NFS를 리용한다. 여기에서는 BSD체계에 대하여 설명한다(BSD체계는 HP-UX, IRIX, SCO 등와 같은 주콤퓨터에서 Linux만큼 널리 사용되고 있다).

의뢰기가 설치요구를 받으면 mountd는 의뢰기의 접근권한을 확인하기 위하여 봉사기의 /etc/exports파일을 검사한다. 이 파일은 원격으로 설치될수 있는 반출된 모든 등록부에 대한 행을 포함한다. 또한 허가된 접근의 형태와 등록부를 설치하도록 인증된 주콤퓨터를 지정할수 있다. 아래에 /etc/exports의 전형적인 몇개의 항목을 주었다.

```
/
/project3/doc
/java/programs   -ro
/hrd/html  -access=fredo:tessio
/prog/html  -rw=michael
```

Linux에서의 형식은 좀 다르다. 등록부이름뒤에 공백으로 구분된 하나이상의 주콤퓨터의 목록을 준다. 모든 주콤퓨터이름에는 괄호안에 허가권의 형태가 있다. 첫 두 행은 이 체계에서도 같으므로 보여 주지 않는다.

```
/java/programs  (ro)
/hrd/html  fredo(rw) tessio(rw)
/projects       *.planets.com(rw)
```

첫행은 모든 기계에 대하여 읽기전용허가권을 지정하며 두번째 행은 주콤퓨터 fredo와 tessio에 읽기쓰기접근을 제공한다. 마지막행은 planets.com령역안의 모든 주콤퓨터들에 읽기쓰기접근을 허용한다.

일단 등록부가 태워 지면 nfsd에 의하여 접근이 조종된다. mountd와 충분한 nfsd프로쎄스가 동작하고 있는가를 확인하고 그다음 exportfs지령으로 /etc/exports안의 등록부들을 반출한다.

```
exportfs -a
```

관리자의 체계가 exportfs지령을 가지고 있지 않다면 mountd와 nfsd데몬들을 개시하는 명령문들을 포함하는 스크립트를 배치해야 한다. 이 스크립트가 restart기능을 가지고 있지 않다면 그때는 그 데몬을 정지시키고 그 스크립트에 인수로서 이 단어들을 제공한 다음 다시 시동해야 한다.

일단 봉사기가 준비되면 의뢰기는 mount지령을 실행하여야 한다. 이때 파일체계형을 nfs로 지정해야 한다. 읽기전용방식으로 배경(bg)에서 유연한 태우기(soft)를 해보자.

```
mount -r -F nfs -o soft,bg sunny:/project3/doc /fredo/project3
```

유연한 태우기는 태우기가 실패하는 경우 의뢰기가 그 조작을 반복하지 않게 한다. 문서읽기에서 이 방식을 쓸수 있다. 등록부를 읽기쓰기방식으로 태웠다 해도 mount의 -r선택항목을 쓰면 /etc/exports가 무효로 된다. 사용자는 또한 remount선택항목을 사용할수도 있는데 그것은 다른 mount선택항목에 속하여 이미 설명되였다.

국부파일체계와 같이 NFS파일체계도 /etc/fstab안에 지정될수 있다. 그때는 mount를 그렇게 복잡한 선택항목들을 가지고 사용할 필요가 없다. fstab가 아래의 항목을 포함한다면 우의 태우기동작이 기동시에 자동적으로 수행될수 있다.

```
sunny:/project3/doc /fredo/project3 nfs ro,soft,bg 0 0
```

NFS는 망을 통하여 파일접근을 공유하는 가장 일반적인 방법이다. NFS는 태우기가능한 파일체계상에 한개의 쏘프트웨어를 보관해 두고 자주 쓸수 있다는 점에서 아주 쓸모 있다. 여기서 우점은 그 쏘프트웨어의 갱신이 오직 한 장소에서만 수행된다는것이다. 일부 기관들은 자기들의 우편을 집중화하기 위하여 우편등록부 /var/mail을 공유하는데 NFS를 사용한다.

참고

관리자가 NFS를 사용하여 원격파일체계에 있는 자기의 파일들에 접근하려 한다면 자기의 UID와 GUID가 봉사기의것과 맞는가를 확인해야 한다. 맞지 않다면 ls -l결과목록은 이름이 아니라 번호로서 그것들을 보여 준다. 더우기 관리자는 자기의 파일상에 요구된 접근권한을 가지지 못할수도 있다. 이 경우에는 정확한 UID와 GUID를 얻기 위하여 /etc/passwd를 수정하는것이 더 좋다.

요 약

TCP/IP망의 기능은 주로 2개의 통신규약 즉 전송조종규약(TCP)과 인터네트통신규약(IP)에 의하여 조종된다. TCP는 믿음성 있는 통신규약이다. 그것은 상대측에서 제때에 수신하지 못한 분실된 토막들을 재전송한다. IP는 파케트주소화를 보장하며 필요하다면 가장 가까운 경로기에 파케트를 발송한다.

망의 주소는 몇개의 8비트들로 표시되며 첫 8비트값은 망의 클라스(A, B ,C)를 결정한다. 망주소의 개별적인 묶음이 국부적인 통합망에서의 사용을 위하여 예약되여 있다. 이 주소들은 인터네트상에서 사용되지 않는다.

모든 망은 귀환 및 방송용으로 각각의 IP주소를 요구한다. 부분망마스크는 IP주소를 해석하여 주콤퓨터주소로부터 임의의 비트들을 망주소의 부분을 형성하는데 채용하였는가를 알려 준다.

망대면부기판은 유일적인 MAC주소를 가지며 통보문이 주콤퓨터에 당도하기 위해서는 IP주소가 이 주소로 변화되여야 한다. 대면부를 구성할 때 관리자는 IRQ와 I/O주소를 지정하여야 할수 있는데 그것은 그 기계의 다른 장치들이 사용하는것과 충돌하지 말아야 한다.

ifconfig는 망대면부의 IP주소와 부분망마스크를 설정한다. 그것은 또한 대면부를 능동 및 비능동화하는데 사용될수 있다. ping과 netstat는 망의 접속을 검사하는데 사용된다.

핵심부는 경로조종표를 관리하며 IP가 경로조종을 목적으로 그 표를 감시한다. route는 망에로의 경로를 설정하며 경로조종표를 현시한다. 표는 외부망으로 향한 파케트들의 경로조종에 사용되여야 할 관

문들을 지적한다. 탐색이 실패한 경우 경로조종표로부터 기본경로가 제공된다.

telnet, ftp, pop3과 같은 많은 TCP/IP봉사들은 inetd가 /etc/inetd.conf안의 항목들을 읽어 내여 호출한다. 데몬들은 흔히 직접적으로가 아니라 외피프로그람(tcpd와 같은)에 의하여 시동된다. 이 봉사들이 사용하는 포구번호들은 /etc/services에 들어 있다.

점대점규약(PPP)은 TCP/IP기능을 전화선상에서 리용할수 있게 한다. pppd는 하드웨어흐름조종만을 사용하며 쏘프트웨어흐름조종은 사용하지 않는다. 대부분의 ISP들이 pppd를 사용하여 련결의 량측에서 동적인 IP주소를 제공한다. 기동할 때와 완료전에 pppd는 각각 /etc/ppp/ip-up과 /etc/ppp/ip-down지령을 실행한다.

인터네트에 접속하기 위해서는 /etc/resolv.conf안에 이름봉사기주소들이 지정되여야 한다. dip와 chat는 가입프로쎄스를 자동화하기 위하여 송신문자렬조를 사용하는 스크립트기반의 도구들이다. pppd는 chat스크립트를 불러 내며 dip는 자체의 스크립트로부터 pppd데몬을 개시할수 있다. PPP접속은 netstat -rn과 ping을 가지고 검사하여야 한다.

대부분의 ISP는 사용자를 인증하기 위하여 PAP 또는 CHAP를 사용한다. PAP는 /etc/ppp/pap-secrets안에 저장된 명백한 본문통과암호를 보내지만 CHAP는 도전문자렬과 공유비밀(/etc/ppp/chap-secrets안에 있는것을 사용하여 계산을 진행한다. 그 파일들은 련결의 량측에서 관리된다. CHAP가 PAP보다 훨씬 더 안전하다.

망파일체계(NFS)는 원격파일체계나 등록부를 국부등록부상에 태울수 있게 한다. 접근권한이 포함된 등록부들은 /etc/exports에 들어 있으며 exportfs로써 반출된다. NFS는 사용자준위의 접근을 제공하지 않는다. /etc/fstab, mount, umount는 단독주콤퓨터에서와 거의 같은 방식으로 사용된다.

시험문제

1. 다음의 IP주소를 가지는 망의 클라스를 지정하시오.

   (1) 202.54.9.1 (2) 107.35.45.78 (3)34.67.102.34
2. 관리자는 인터네트상의 주콤퓨터의 IP주소로서 11.23.34.45와 172.26.0.6을 가질수 있는가?
3. 관리자의 기계상에서 TCP/IP를 구성하기전에 수동으로 설정할 필요가 있을수도 있는 망대면부기판의 두가지 하드웨어파라메터들은 어떤것인가?
4. 망대면부기판의 IRQ와 I/O주소를 어떻게 찾아 내는가?
5. ftp봉사를 어떻게 차단하는가?
6. TCP/IP회선접속에서 어느 통신규약이 가장 널리 사용되며 어느 지령으로 그 봉사를 능동으로 만드는가?
7. 어느 Hayes모뎀지령으로 회선접속하는가?

8. /etc/exports는 무엇을 포함하며 그 파일을 변화시킨후에 어떻게 mountd가 이 파일을 읽도록 할수 있는가?

련습문제

1. 기계의 MAC주소를 어떻게 찾아 낼수 있는가?
2. 방송이란 무엇이며 그것은 어느 IP주소를 사용하는가?
3. 귀환주소의 의미는 무엇인가?
4. 관리자가 C클라스인트라네트를 설정하고 있다면 공식적인 지도서를 따를 때 첫 두개의 8비트는 무엇으로 되는가?
5. 스크립트를 사용하지 않고 관리자의 망을 어떻게 부동으로 만드는가?
6. ping이 몇개의 파케트분실을 보여 준다면 그것은 무엇을 가리키는가?
7. 관리자의 기계가 인터네트에 접속된 다른 기계(192.168.0.1)에 련결되였다. 그 관문기계가 관리자로 하여금 인터네트에로 파케트들을 보내는것을 허락한다고 가정할 때 SVR4와 Linux에서 관리자의 기계에 어떻게 경로조종을 설정하겠는가?
8. 관리자의 경로기 IP주소를 어떻게 찾아 내는가?
9. 관리자의 기계상에서 어떻게 telnet봉사를 불가능하게 하겠는가?
10. 권한 없는 사용자등록자리로 /usr/sbin/pppd지령을 실행시킬수 없다면 그렇게 될수 있는 리유는 무엇인가?
11. 관리자는 FQDN이 아니라 IP주소를 사용해서만 인터네트상의 ftp싸이트에 접속할수 있다. 이름봉사기가 관리자의 망에서 구동하고 있는가를 고찰하는데서 무슨 설정을 잊었는가?
12. CHAP인증이 왜 PAP보다 우월한가?
13. 스크립트로부터 인터네트상의 ftp싸이트에 접속할 필요가 있지만 ftp지령을 실행하기전에 netstat -rn출력에 단어 ppp0이 있는가를 확인해야 한다. 그것을 어떻게 할것인가?
14. 한 행의 perl프로그람을 사용하여 자기의 모든 .chat와 .dip스크립트의 통과암호를 s1o3n5y8로부터 j2n98d0k2까지 변화시키시오.
15. 주콤퓨터 uranus에 있는 원격등록부 /usr/doc를 읽기전용방식으로 접근하려면 SVR4체계상에서 mount를 어떻게 사용하겠는가?

제 24 장. 인터네트봉사기의 구축

이 장에서는 UNIX를 리용하여 현재 널리 알려 진 인터네트봉사에 대하여 설명한다. UNIX는 인터네트봉사에서 오래동안 주되는 역할을 해왔다. 최근까지도 모든 인터네트봉사기들은 현실적으로 UNIX기계였다. 그 기술이 비독립적이였고 개방규격에 기초하고 있었지만 그래도 UNIX는 이 령역을 지배하였다.

여기서는 인터네트의 중추를 이루는 중요한 3가지 봉사 즉 DNS(이름봉사), 우편봉사, Web봉사를 설명한다. 그리고 이 3가지 봉사들이 어떻게 작업하는가를 리해할뿐아니라 자기 손으로 이 봉사들을 설치할수 있도록 이끌어 준다. 여기서 설명되는 내용은 임의의 UNIX체계에 적용될수 있다. Linux에서 모든 인터네트봉사가 자유로와 졌으므로 Linux를 사용하여 인터네트봉사기를 구성하여 보도록 한다.

명백하면서도 확장성이 있는 방식으로 인터네트봉사기를 구성하기 위하여 실례로서 가상적인 령역이름을 가진 어떤 회사들을 리용한다.

이 장에서는 다음과 같은 내용들을 학습하게 된다.

- 우편과 Web봉사를 위한 어떤 기관의 요구를 파악한다(24.1).
- 지역자료를 관리하기 위하여 DNS가 어떻게 이름봉사를 사용하는가를 배운다(24.2).
- 주이름봉사기를 구성한다(24.3).
- 분석기를 구성하고 named데몬을 관리한다(24.5, 24.6).
- sendmail이 어떻게 우편을 보내고 받는가를 배운다(24.7).
- /etc/sendmail.cf를 리용하여 몇가지 공통적인 요구를 처리한다(24.8).
- 별명을 리용하여 우편을 발송한다(24.9).
- 인터네트에 접속된 주콤퓨터와 회선을 사용하는 주콤퓨터를 위하여 우편봉사기를 설치한다(24.10).
- fetchmail을 리용하여 POP봉사기로부터 우편을 내리적재한다(24.11).
- HTTP가 어떻게 작업하는가를 리해하며 httpd.conf를 구성하여 몇가지 공통적인 Web봉사기기능을 관리한다(24.13, 24.14).
- 가상주콤퓨터를 설치하고 등록부접근을 조종하는데 사용되는 기술을 파악한다(24.14.6, 24.15).

24.1 실례망(중심국과 위성국)

Rational Planets회사(가상적인 회사:중심국이라고 하자.)는 임대 받은 선로를 통하여 인터네트에 직접 접속된 넓은 망을 가지고 있다. 이 회사는 령역이름 planets.com을 할당 받았으며 자체의 망에 인터네트봉사를 수립할 계획이다. 이 망에서 주콤퓨터 saturn과 uranus는 이름봉사를 수행하도록 하며 jupiter와 saturn은 우편봉사를 주관하도록 한다. 그중에서 saturn과 jupiter는 개별적인 부류에서 봉사기로서의 위치를 차지한다. neptune은 Web봉사를 주관한다.

망의 나머지콤퓨터들은 **방화벽**(firewall) 즉 외부의 공격으로부터 내부망을 보호하는 콤퓨터의 뒤에 배치된다. jupiter는 방화벽으로서 동작하며 내부망의 일부 기계들이 이 관문을 통하여 인터네트에 접근

하도록 하는 전송기능을 제공한다. 이러한 워크스테이션들의 일부는 Windows를 사용하고 밤에는 스위치를 끄며 jupiter로부터 자기들의 우편을 내리적재하여야 한다. 몇대의 워크스테이션은 Linux를 사용하고 시종일관 가동하며 jupiter로부터 우편들이 자동적으로 전송된다. 한명의 사용자가 회선을 사용하여 집에 있는 자기의 기계 mercury로부터 망에 접속한다.

Rational Planets는 아주 큰 설치를 가지므로 자기의 일부 공간과 봉사를 다른 기관에 임대하기로 결정하였다. Rational Velvet(Rational Planets보다 작은 기관:위성국이라고 하자.)는 velvet.com이라는 령역이름을 받았다. Velvet는 인터네트에 직접 접속하지 못하므로 Planets에서 쓰이는 기능들을 리용한다. Planets는 Web봉사를 주관하기 위한 20MB와 그들을 대신하여 모든 우편을 받기 위한 20MB의 디스크공간을 Velvet에 제공할것이다. jupiter는 Velvet를 위한 우편도 받아 들이며 몇개의 FQDN 즉 jupiter.planets.com, mail.planets.com, mail.velvet.com, velvet.com을 가진다.

Planets는 20MB의 한계내에서 Velvet에 무수히 많은 우편통들을 제공한다. 전자우편통보문은 user@velvet.com을 사용하여 Velvet내의 임의의 사용자에게 보내여 질수 있다. 마찬가지로 user@planets.com으로써 Planets내의 사용자들을 지정한다. Planets의 싸이트로부터 우편을 회수할수 있도록 하기 위하여 Velvet는 FQDN scarletisp.com을 가지고 있는 국부ISP와 함께 참가하였다. 이 ISP는 또한 Velvet에 빈 전자우편등록자리 velvet@scarletisp.com도 제공한다. Velvet는 PPP를 사용하여 ISP에 도간도간 접속하고 두개의 원천 즉 자기 령역에서 Planets의 봉사기와 ISP에서 관리되는 하나의 전자우편통으로부터 우편을 내리적재한다.

얼마후에 Velvet는 Planets가 그들에게 자기의 Web싸이트를 주관하도록 배당한 20MB를 사용하기로 결정한다. 주콤퓨터 neptune은 3개의 FQDN 즉 neptune.planets.com, www.planets.com, www.velvet.com을 가진다. 이제 이 가설의 대부분을 실현하자. 먼저 그림 24-1을 보시오.

24.2 령역이름봉사

인터네트는 주콤퓨터이름을 주소로 해석하는데서 /etc/hosts가 아니라 령역이름봉사(Domain Name Service: DNS)를 사용하는데 이것은 주콤퓨터이름-주소넘기기들의 거대한 분산자료기지를 리용한다. 이 체계는 파울 모카페트리스(Paul Mockapetris)에 의하여 발단되였으며 그가 DNS의 특성을 서술하고 처음으로 실현하였다. 그러나 그후에 케빈 던래프(Kevin Dunlap)는 버클리UNIX에서 BIND(Berkeley Internet Name Domain)라고 불리우는 훨씬 통속적인 실현을 서술하였다. BIND 8은 지금 모든 UNIX 기계들에 보급되고 있다.

BIND는 주콤퓨터이름-주소자료기지를 가지고 있는것외에 어떤 령역에 대한 우편을 취급하는 봉사도 한다. henry@neptune.planets.com으로 주소화된 통보문은 주콤퓨터 neptune안에 반드시 도착하지 않아도 된다. BIND는 하나이상의 **우편교환기**(mail exchager: 우편을 받는 주콤퓨터)에 순서를 지정하여 주콤퓨터들이 옳바른 순서에 따라 시도되도록 한다. 더우기 BIND가 FQDN을 분석할수 없다면 같은 봉사를 실행하고 있는 몇개의 다른 주콤퓨터들에 그 문제를 부탁할수 있어야 한다.

DNS는 어떻게 동작하는가

DNS는 령역의 이름공간을 대응하는 권한을 받은 **지역**(zone)들로 나눈다. 매 지역의 관리자는 하나이상의 **이름봉사기**(name server)(그 지역의 이름-주소정보를 포함하는 자료기지)들을 관리할 의무를 가지고 있다. **주(1차)이름봉사기**(master name server)는 최종정보를 포함한다. **종속(2차)이름봉사기**(slave

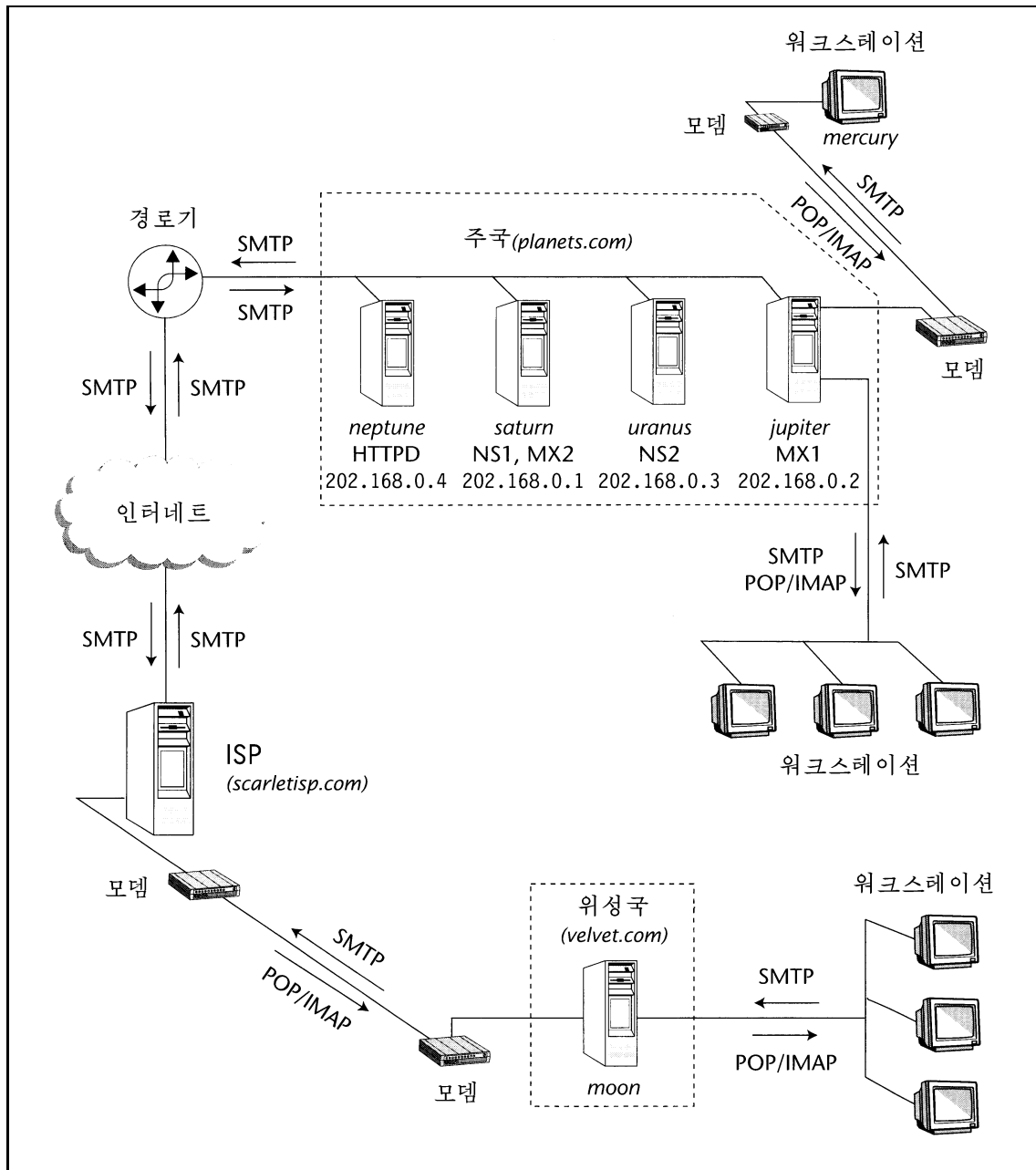



그림 24-1. 실례 Rational Planets의 망

name server)는 **지역전송**(zonal transfer)을 통하여 주이름봉사기로부터 정보를 얻는다. 종속이름봉사기 또한 기본봉사기의 실패사건시에 여벌복사체계도 봉사한다. 3번째 형식의 이름봉사기 즉 **완충전용봉사기**(caching-only)가 있는데 이것은 질문들을 단순히 13개의 기본이름봉사기들의 특정한 묶음에로 돌려 준다. 3가지 형태의 설치을 모두 고찰한다.

령역과 지역의 차이점은 아주 미묘하지만 리해하여야 한다. 령역 birds.edu가 더 나아가서 2개의 부분령역 parrots.birds.edu와 cuckoo.birds.edu로 나누어 진다고 하자. birds.edu의 관리자는 cuckoo.birds.edu의 관리를 위임하는 한편 parrots.birds.edu의 관리는 유지하기로 결정하였다. 여기에 명확한 2개의 지역 즉 cuckoo.birds.edu와 birds.edu가 있다. 여기서 birds.edu지역은 부분령역 parrots.birds.edu를 포함하지만 cuckoo.birds.edu는 포함하지 않는다. 하나의 파일체계를 형성하기 위하여 여러개의 파일체계들이 결합되는것과 같은 방식으로 지역들이 모여 전반적인 령역의 통합된 형상을 보여준다.

모든 지역은 이름봉사기들(주 및 종속)의 묶음을 가지며 그것들이 제공하는 대답은 그 지역에 대하여서는 **권한이 있는**(authoritative)것이다(정당한것이다). 이름봉사기는 **분석기**(resolver)로부터 질문을 받으며 분석기는 응용프로그람을 대신하여 주콤퓨터의 IP주소를 얻는다. 그것은 독자적인 프로그람이 아니라 telnet나 ftp와 같은 응용프로그람들에 련결되는 서고루틴들의 묶음이다. 보통 IP주소를 얻기 위해서는 여러개의 이름봉사기들에 질문을 보내야 한다. 이것은 이름봉사기가 수행할 일감이며 따라서 분석기는 모든 작업을 국부이름봉사기에 의존한다.

질문이 제기되면 이름봉사기가 그 분석을 수행할수도 있지만 그렇지 않다면 포기하는것이 아니라 희망하는 주콤퓨터에로 한 걸음 더 접근시켜 줄 또 하나의 이름봉사기의 IP주소를 제공할수 있게 설계되여 있다. 만일 그 봉사기도 둘중의 어느 대답도 제공할수 없다면 여전히 또 다른 봉사기에 이 문제를 부탁해야 한다. DNS에서 의뢰기-봉사기방식은 주소가 최종적으로 분석될 때까지 그 부탁들과 지역들사이의 련결이 제대로 개척되도록 확인한다.

우의 birds.edu령역에서 birds.edu지역의 이름봉사기에 cuckoo.birds.edu지역에 있는 주콤퓨터의 IP주소를 문의한다면 분명 대답이 없다. 그러나 cuckoo.birds.edu지역에 있는 이름봉사기(봉사기는 대답을 안다.)의 IP주소를 제공할수 있어야 한다.

분석(resolution)은 계층구조로 되여 있다. 만일 국부봉사기가 www.cuckoo.birds.edu(여기서 www는 주콤퓨터이다.)의 주소를 제공할수 없다면 그 봉사기는 cuckoo.birds.edu나 birds.edu 등의 이름봉사기를 알고 있는가를 자기의 레코드들로부터 검사한다. 어떤 점에서 이것은 정지되여야 하며 그 전진은 **뿌리이름봉사기**(root name server)에서 정지된다. 이 봉사기들은 com, edu, ca, gb 등과 같은 모든 웃준위령역들의 이름봉사기들을 알고 있다. 인터네트상의 모든 이름봉사기들은 뿌리이름봉사기들의 IP주소를 가지고 있으며 모든 분석이 실패하는 경우는 직접 그것들과 접촉한다.

어느 한 주콤퓨터가 cuckoo.birds.edu의 국부이름봉사기에게 www.planets.com의 주소를 알고 있는가를 문의한다고 가정하자. 그것은 명백히 대답을 몰라 뿌리이름봉사기들을 직접 접촉한다. 그것들중 하나가 com이름봉사기의 IP주소를 돌려 준다. 국부이름봉사기는 이제는 com이름봉사기에 문의하며 그다음 planets.com이름봉사기에 그것을 부탁한다. 이 봉사기는 www.planets.com의 주소를 알고 있으며 대답을 돌려 준다. 매 뿌리이름봉사기는 초당 수천개의 질문을 처리한다.

주해

뿌리이름봉사기들의 일부는 웃준위령역의 이름봉사기들의 IP주소를 제공해 줄뿐아니라 실제적으로 이 령역들의 이름봉사기들이다.

24.3 주봉사기의 구성

UNIX에서의 이름봉사기인 BIND는 named데몬에 의하여 조종된다. ps -e(Linux에서는ps ax)지령이 named가 실행하고 있다는것을 보여 주며 named가 실행되고 있다면 BIND가 관리자의 체계안에서 실행하고 있다는것이다. 최종판본은 BIND 8이며 이전에 사용되였던 BIND 4보다 4만큼이나 더 큰 판번호를 사용한다. named는 어느 한 rc스크립트로부터 실행되지만 쉘스크립트 ndc도 역시 쓸모가 많다. 관리자의 BIND구성을 시험하여 보면 ndc가 아주 편리하다는것을 알게 된다.

이 체계는 구성파일 /etc/named.conf(BIND 4에서는 /etc/named.boot)를 사용하며 주콤퓨터이름-주소넘기기를 포함하는 2~4개의 추가적인 파일을 사용한다. 매 봉사기형태 즉 주봉사기, 종속봉사기, 완충전용봉사기를 위하여 BIND 8을 구성한다.

/etc/named.conf(named에 의하여 사용되는 기본파일)는 파일체계의 상위블로크와 류사하다. 그것은 그 기계에 의하여 관리되는 봉사기의 형태나 지역들(봉사기는 바로 그 지역에 대한 레코드들을 처리한다.)과 자료기지파일의 이름 및 위치와 같은 대략적인 정보들을 포함한다. named는 이름봉사기의 형태에 따라 아래와 같은 파일들을 사용할수 있다.

- 뿌리이름봉사기의 IP주소와 FQDN을 포함하는 암시파일(hints file): named는 항상 기억기에 이 봉사기들의 목록을 보존하므로 주콤퓨터이름을 분석할수 없을 때 그것들을 사용할수 있다.
- 귀환주소 127.0.0.1을 모조콤퓨터이름 localhost로 변환하는 국부주콤퓨터파일(localhost file)
- 이 봉사기에 의해 봉사되는 지역의 모든 주콤퓨터들의 기본자료기지를 포함하는 지역파일(zone file): 이 파일은 또한 그 지역의 이름봉사기들과 우편봉사기들의 IP주소도 포함한다. 매 지역은 별개의 지역파일을 가진다.
- IP주소를 대응하는 주콤퓨터이름으로 바꾸는 역탐색파일(reverse lookup file): 일부 응용프로그람(rsh와 같은)들이 이 변환을 수행할 필요가 있다. 매 지역은 자기의 역탐색파일을 가지고 있다.

주봉사기는 4개의 파일을 모두 요구하며 종속봉사기는 암시파일과 국부주콤퓨터파일만을 가지고 있어야 한다. 종속봉사기는 지역을 만들고 주봉사기로부터 역탐색파일들을 적재한다. 완충전용봉사기는 암시파일을 필요로 하지만 보통 국부주콤퓨터파일도 가진다. 망주소 202.168.0을 가지는 령역 planets.com에서 매 이름봉사기들을 구성해 보자.

인터네트에 직접 접속된 망들에서 어느 한 주콤퓨터를 주봉사기로 쓸 필요가 있다. 먼저 4개의 파일을 만들고 그것들을 별개의 등록부에 넣은 다음 파일들과 그 위치를 지정하는 구성파일(named.conf)을 만든다. 기본구성파일 /etc/named.conf를 제외한 다른 파일들은 임의의 이름을 가질수 있다. 여기서는 saturn이 주봉사기로서 동작한다.

24.3.1 암시파일

암시파일(hints file)은 뿌리이름봉사기들의 목록을 named가 리해할수 있는 형식으로 포함한다. 인터네트상에서 암시파일은 13개이며 그것들은 모든 웃준위령역의 이름봉사기주소들을 제공한다. 만일 netscape.com의 이름봉사기의 IP주소를 알려고 한다면 암시파일을 쓸수 있다. 암시파일을 named.cache라고 부르겠다.

```
# cat named.cache
.                       3600000  IN  NS   A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A    198.41.0.4
.                       3600000      NS   B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000      A    128.9.0.107
.                       3600000      NS   C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.     3600000      A    192.33.4.12
.....
.                       3600000      NS   M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.     3600000      A    202.12.27.33
```

우의 내용은 4개의 뿌리이름봉사기들을 위한 항목을 보여 주기 위하여 편집되였다. named는 시동할 때 이 파일을 읽고 거기에 렬거된 뿌리이름봉사기의 하나에 접촉하여 최종파일을 얻는다. 그때 이 목록

을 자기의 기억기에 보존한다. 관리자는 주기적으로 ftp://ftp.internic.net/domain/named.root로부터 최종파일을 얻어야 한다. 관리자는 이 파일을 편집하지 말아야 하며 named가 리해할수 있는 형식으로 항상 유지하여야 한다.

모든 뿌리봉사기에서 2개의 기록 즉 NS와 A가 있는데 그것들을 간단히 설명하도록 하자. A레코드는 봉사기의 IP주소와 FQDN이 들어 있다. 이것은 DNS가 주소넘기기를 어떻게 관리하는가 하는것이며 지역파일에서 그 형식이 반복되는것을 본다.

주해

Linux기계상에서 /var/named/named.ca 또는 /var/name d/root.hint로서 자기 몫에 해당한 암시파일을 찾아 볼수 있다. 원한다면 그 이름을 바꿀수 있다.

24.3.2 국부주콤퓨터파일

국부주콤퓨터파일은 그 기계의 귀환주소의 분석을 수행하는데 필요한 정보를 제공한다. named는 주소 127.0.0.1을 이름 localhost로 바꿀수 있어야 한다. 이 파일을 planets.local이라고 부르겠다.

```
# cat planets.local
@    IN     SOA     localhost.     root.localhost. (
                    20000061601    ; Serial number
                    28800          ; Refresh
                    1400           ; Retry
                    604800         ; Expire
                    86400)         ; TTL
     IN     NS      localhost.
1    IN     PTR     localhost.
```

이 파일형식은 주봉사기에서 사용하는 2개의 다른 파일들이 공유하며 따라서 관리자는 그것이 구조화되는 방식을 리해하여야 한다. 그 파일은 SOA(Start of Authority)레코드로 시작되는데 그 레코드는 행의 첫 문자로서 @를 가진다. 이 @는 현재의 원천(origin) 즉 이 파일에 의하여 표시되는 지역을 가리킨다. 이 경우에 이 원천은 planets.com지역이다. 그뒤에는 예약어 IN(Internet)과 SOA가 놓이며 뒤이어 주콤퓨터이름(localhost)과 우편을 받을 사용자의 전자우편주소(root.localhost.)가 놓인다.

여기서 주의해야 할 두가지 문제가 있다. 마지막의 두 이름은 점(.)으로 끝난다.일반적으로 관리자는 이 구성파일에서 사용되는 주콤퓨터의 FQDN이 점으로 끝나는가를 확인해야 한다. 그렇지 않으면 named가 그것을 주콤퓨터이름으로 리해하고 령역이름을 첨부하려고 한다. 또한 @가 named안에서 원천을 가리키기때문에 전자우편주소가 root@localhost가 아니라 root.localhost로서 표현된다는것도 주의해야 한다.

SOA레코드는 괄호로 닫긴 5개의 마당을 가진다(반두점뒤에 보여 준 설명을 참고). 계렬번호는 이 파일이 변화될 때마다 갱신되여야 한다. 종속봉사기는 자료기지를 다시 끌어 와야 하겠는가를 결정하기 위하여 주봉사기의 파일들에서 이 항목을 주기적으로 검사한다. 관리자는 간단한 계렬번호나 YYYYMMDDNN형식(NN은 련번호를 가리킨다.)을 가진 날자형식의 번호를 가질수 있다.

재생주기(refresh cycle)는 종속봉사기가 지역전송을 해야 하겠는가를 결정하기 위하여 22800초마다 계렬번호를 검사하게 한다. 이 재생주기는 이것보다 작은 값을 가질수 있어도 큰 값은 가질수 없다. 관리자는 변화가 일어 났다는것을 종속봉사기에 알리는 notify문을 /etc/named.conf안에 넣을수 있다.

만일 주봉사기가 응답을 무시하면 종속봉사기는 1400초간격으로 자료를 가져 오려고 시도한다. 주봉사기가 정지되여 있거나 종속봉사기가 그 파일로 갱신할수 없다면 또다시 604800초(7일간)동안 이름봉사기질문들에 응답할것을 계속 요구한다. TTL(Time-to-Live)마당은 이 마당이 지정되지 않은 모든 레코드들이 사용할 기정값을 보여 준다. 여기서는 하루로 설정된다.

SOA레코드의 뒤에 2개의 추가적인 레코드가 있다. 처음것은 그 지역의 이름봉사기로서 국부주콤퓨터를 보여 주는 NS(name server)레코드이고 다음것은 PTR(pointer)레코드(이 레코드를 0.0.127.in.addr.arpa에 대한 역분석을 하기 위한 레코드로서 접수한다.)이다. PTR(pointer)레코드는 후에 설명한다. 여기서 주소 1(즉 127.0.0.1)을 가진 주콤퓨터는 이름 localhost로 변환된다.

24.3.3 지역파일

지역파일은 모든 주콤퓨터이름-주소넘기기들이 보존되여 있는 곳이다. 즉 /etc/hosts의 기능의 대부분을 교체한것이다. 지역 planets.com을 위하여 지역파일을 따로 가지고 있어야 한다. 이 파일은 비트조합(그림 24-2)이며 따라서 그의 해체에 착수하기전에 한번 보기로 하자.

```
@      IN      SOA     saturn.planets.com.   root.saturn.planets.com. (
                               3           ; Serial number
                               28800       ; Refresh
                               1200        ; Retry
                               604800      ; Expire
                               86400)      ; TTL
; Name servers and mail servers
               IN        NS      saturn.planets.com.
               IN        NS      uranus.planets.com.
               IN        MX  10   jupiter.planets.com.
               IN        MX  20   saturn.planets.com.
; Hosts of this zone
localhost      IN        A       127.0.0.1
saturn         IN        A       202.168.0.1
jupiter        IN        A       202.168.0.2
uranus         IN        A       202.168.0.3
planets.com.   IN        A       202.168.0.3
neptune        IN        A       202.168.0.4
               IN        MX  10   jupiter.planets.com.
; Aliases
mail           IN        CNAME    jupiter.planets.com.
www            IN        CNAME    neptune.planets.com.
ftp            IN        CNAME    uranus.planets.com.
```

그림 24-2. 지역파일 planets.master

여기서 파일은 3개의 부분으로 나누어 지며 중간규모의 망에서 주이름봉사기를 설정할 때 알아 둘 필요가 있는 모든 인자들을 포함하고 있다. SOA레코드는 주봉사기의 FQDN을 saturn.planets.com으로서 기록하며 BIND와 관계되는 모든 전자우편은 root@saturn.planets.com으로 주소가 지정되여야 한다. 계렬번호 및 시간과 관계되는 파라메터들은 이미 설명되였다.

지역파일은 **자원레코드**(RR:Resource Record)들의 존재에 의하여 특징 지어 지며 이미 그것들의

일부를 보았다. 매 자원레코드들은 4~5개의 마당을 가진다. 레코드형은 3번째 마당에 있으며 IN은 항상 2번째 마당에 있다. 첫번째와 마지막마당에는 주콤퓨터이름과 IP주소가 있다. 여기서 RR의 4가지 형태를 보기로 하자.

- NS: 이름봉사기레코드. 이 레코드는 주봉사기와 종속봉사기의 위치를 가리킨다.
- A: 주소레코드. 이 레코드에는 주콤퓨터이름-주소넘기기가 /etc/hosts에 비하여 좀 확장된 형식으로 들어 있다.
- MX: 우편교환기레코드. DNS는 령역에 대한 우편을 수신하도록 되여 있는 봉사기들도 지정한다.
- CNAME: 별명(주콤퓨터의 **본명**(canonical name)대신에 사용할수 있다. 실례로 neptune을 www.planets.com으로서 접근한다고 해도 www.planets.com에 대한 본명은 neptune.planets.com으로 된다.

여기에 2개의 이름봉사기(NS)레코드가 있다. 첫번째것은 saturn.planets.com에 대한 주봉사기이고 다음것은 uranus.planests.com에 대한 종속봉사기이다. 이 이름봉사기들은 자체로 FQDN을 완성하므로 그뒤에 점이 놓여야 한다. 그것들에는 둘 다 첫 마당이 없다. 이 마당은 선택적이며 제공하지 않을 때에 named는 주소화되는 마지막지역(이 경우에는 현재의 지역 planets.com)을 표현하는것으로 리해한다.

다음으로 우편교환기레코드(MX)가 있다. jupiter는 우선권 10을 가지는 가장 우선권이 높은 우편봉사기이다. 주이름봉사기 saturn은 여벌우편봉사기(backup mail server)이기도 하지만 더 낮은 우선권(20)을 가지고 있다. named는 planets.com지역에 들어 오는 모든 우편을 jupiter에게 보내며 실패하면 saturn이 접수한다. 만일 그 워크스테이션이 인터네트에 대한 접근을 가지고 있지 않다면 망관리자는 jupiter가 정지되는 경우에 saturn과 내부부분망사이의 접속을 설정해야 한다.

다음부분은 왼쪽에 주콤퓨터이름, 오른쪽에 IP주소를 가지는 자료기지를 포함한다. 여기서 거의 모든 주콤퓨터이름들은 점을 가지고 있지 않으며 이것은 named가 주소를 찾아 보기전에 령역이름을 추가한다는것을 의미한다. 그것들중 하나(neptune)는 별개의 MX레코드도 포함하고 있다. BIND가 이것을 허락한다는것은 user@neptune.planets.com 으로 주소가 지정된 모든 우편이 jupiter에게도 접수된다는것을 의미한다. jupiter는 또한 user@planets.com형식으로 우편을 받아 들인다.

관리자는 주콤퓨터이름들이 별명을 가지도록 하려고 할것이며 BLND는 CNAME레코드를 통하여 그것을 제공한다. 이 레코드는 왼쪽에는 별명을, 오른쪽에는 그 주콤퓨터의 본명을 가진다. 그것은 관리자가 neptune상에 관리되는 Web싸이트에 접근하기 위해 www.planets.com을 사용하게 한다. 여기서 www는 단순히 주콤퓨터 neptune의 별명으로서 동작한다. 사용자는 A레코드를 가지고도 별명을 만들수 있으나 CNAME은 자기항목을 수정함이 없이 jupiter의 IP주소를 변화시킬수 있게 한다는 우점이 있다.

주콤퓨터이름이 없이 령역이름을 보여 주는 특이한 항목(planets.com)이 하나 있다. 이것은 그자체만으로 완전한 FQDN을 이루고 있으므로 점으로 끝나야 한다. planets가 주콤퓨터이름이 아닌데도 관리자는 uranus.planets.com과 ftp.planets.com외에 또 planets.com으로서 주콤퓨터 uranus에 접근할수 있다. 이것은 인터네트상에서 자주 하고 있는것이다. 즉 www.internic.net외에 internic.net도 사용하고 있다.

주의

CNAME레코드가 또 다른 CNAME레코드를 지정하게 하는것은 좋지 않다. 그것은 A레코드만을 지정하여야 한다. 또한 MX레코드는 CNAME(별명)이 아니라 주콤퓨터의 본명을 지정하여야 한다.

24.3.4 역지역파일

역지역파일(reverse zone file)은 역분석(reverse resolution) 즉 IP주소를 령역이름으로 변환하기 위해 요구되는 파일이다. 많은 UNIX도구들이 역분석을 해야 하며 그것을 위한 특수한 지역파일이 있다. named는 모든 역분석을 in-addr.arpa령역에서 수행한다. planets.com을 위한 역령역은 0.168.202.in-addr.arpa(IP8비트들을 뒤집은것)로써 취급된다.

령역이름과 IP주소들이 정반대의 주소를 가지므로 이 8비트들을 거꾸로 놓는것이 마땅하다. IP주소에서 왼쪽끝의 8번째 비트가 가장 큰 의미를 가지는 한편 오른쪽끝의 령역에서도 그것은 같다. IP주소들을 뒤집은 구조가 령역이름들이다. 파일 planets.reverse가 반대항목이다.

```
# cat planets.reverse
@      IN        SOA    saturn.planets.com.   root.planets.com. (
                                  1           ; Serial number
                                  28800       ; Refresh
                                  7200        ; Retry
                                  604800      ; Expire
                                  86400)      ; TTL
            IN     NS     saturn.planets.com.
            IN     NS     uranus.planets.com.
1           IN     PTR    saturn.planets.com.
2           IN     PTR    jupiter.planets.com.
3           IN     PTR    uranus.planets.com.
4           IN     PTR    neptune.planets.com.
```

이 파일은 0.168.202.in-addr.arpa령역을 위한 항목들을 보여 준다. NS레코드와 달리 이 파일은 역분석을 수행하기 위한 지시자레코드(PTR)를 가진다. 여기서 C클라스망을 취급하고 있으므로 IP주소의 마지막 8자리수는 PTR레코드들의 첫렬에 있다.

주해

역분석파일에서 PTR레코드들의 첫 렬은 사실상 IP주소의 주콤퓨터부분을 뒤집은 형식이다. 만일 관리자가 B클라스의 망 152.167.0.0을 가지고 있다면 관리자의 역령역은 167.152. in-addr.arpa이다. IP주소 152.167.34.56을 가진 주콤퓨터는 이 파일에서 첫렬로 56.34를 가진다.

24.3.5 기본구성파일 named.conf

필요한 모든 파일들을 만든 다음 일람파일 /etc/named.conf를 만들어야 한다. 이 파일은 자료기지파일들의 위치와 그들의 봉사목적을 지정한다. 이 파일의 구조는 BIND 4와 BIND 8사이에서 상당히 변화되였다. 그림 24-3에서는 BIND 8판본을 보여 준다.

파일은 명령문들의 묶음으로 되여 있는데 매 명령문들은 앞뒤에 대괄호를 주고 반두점으로 끝낸다. 파일은 봉사기가 조작하는 매 지역에서 한개의 zone문을 포함하며 options문으로 시작한다. 이 명령문은 단순히 모든 지역파일들이 /var/named에 위치한다는것을 나타낸다. 원한다면 다른 등록부이름을 선택할수 있으나 구태여 그렇게 할 필요는 없다. 설명문들은 C++와 같이 앞에 두개의 사선(//)을 붙여 만든다.

```
options {
      directory "/var/named" ;
};
zone "." {
      type hint ;            // used to be specified w/ "cache"
      file "named.cache" ;
};
zone "planets.com" {
      type master ;          // what used to be called "primary"
      file "planets.master" ;
};
zone "0.168.202.in-addr.arpa" {
      type master ;
      file "planets.reverse" ;
};
zone "0.0.127.in-addr.arpa" {
      type master ;
      file "planets.local" ;
};
```

그림 24-3. /etc/named.conf(BIND 8판본)

여기서 설명한 매 파일은 named로 리해한것으로서 지역에 소속된다. 뿌리(.)와 planets.com지역은 정분석(forward resolution)이며 in-addr.arpa령역을 사용하고 있는것들은 역분석(reverse resolution)이다. 완충기파일(cache file)을 제외한 모든 파일들은 주봉사기형식의 파일이며 뿌리이름봉사기들의 IP 주소를 기계의 기억기에로 읽어 들인다. 그 파일들의 위치는 options명령문에서 지정된것처럼 /var/named등록부와 련계하여 보여 진다. 이 구성은 거의 모든 목적에 부합되므로 여기서는 더 상세히 들어 가거나 유용한 다른 선택항목들을 설명하지 않는다.

24.4 보조봉사기와 완충봉사기

saturn에 대한 주봉사기의 지역파일은 uranus를 종속봉사기로서 정의한다. 기본봉사기와 종속봉사기의 구성은 대체로 류사하다. 종속봉사기도 기억기를 가지며 동일한 내용을 가진 국부주콤퓨터파일들을 가진다. 그렇지만 종속봉사기는 주봉사기로부터 읽어 들여 지역파일과 역지역파일을 만든다. uranus의 /etc/named.conf에서 정분석 및 역분석을 위한 두 항목은 아래와 같다.

```
zone "planets.com" {
     type slave;              현재 slave(종속봉사기)이다
       file "planets.slave";             이 파일이 만들어 진다
     master { 202.168.0.1 };             여기로부터 적재된 다음에
};
zone "0.168.202.in-addr.arpa" {
     type slave ;              //우와 같은 설명문
     file "planets.slave.reverse";
     master { 202.168.0.1 };
```

```
};
```

여기에 두가지 차이점이 있다. type는 slave이며 주봉사기로부터 지역파일들을 적재하기 위하여 master문이 사용된다. 종속봉사기는 주봉사기의 IP주소를 알아야 한다. 이 지역전송에 의하여 보조봉사기안에 planets.slave와 planets.slave.reverse파일이 만들어 진다. 이 파일의 내용을 시험하는것으로써 종속봉사기가 제대로 동작하고 있는가를 알수 있다.

우의 파일들은 slave형으로 정의되였지만 역령역 0.0.127.in-addr.arpa를 위한 파일은 여전히 master형이다. 암시파일들을 위한 options문과 zone문은 변하지 않는다.

완충전용봉사기의 구성은 더 간단하다. 그것은 바로 암시파일을 사용한다(때로는 국부주콤퓨터파일도). 따라서 /etc/named.conf안에 이 파일을 가리키는 한개의 zone명령문이 있다. options명령문은 다 같다.

24.5 분석기의 구성

UNIX체계들에서 분석기(resolver)는 구성파일 /etc/resolv.conf를 사용한다. 이 파일은 앞에서 인터네트에 접속하기 위하여 주콤퓨터를 설정할 때 소개되였다. planets.com망에서 모든 주콤퓨터들은 자기의 resolv.conf안에 아래의 항목을 가진다.

```
search planets.com
nameserver 202.168.0.1
nameserver 202.168.0.3
```

nameserver항목은 주콤퓨터 saturn과 uranus를 가리킨다. 해석기는 먼저 이름봉사기를 시도해 보고 그다음 두번째것을 시도한다. 여기서 search명령문을 리용하여 별명과 짧은 이름을 달수 있다. 만일 관리자가 점을 가지지 않는 이름으로 ftp나 telnet을 사용한다면 search행에 지정된 령역들이(여기서는 하나뿐이지만) 그 이름에 추가되며 DNS이름공간이 다시 검색된다. 이것은 ftp uranus.planets.com외에 ftp uranus도 사용할수 있다는것을 의미한다.

Linux체계는 또 다른 파일 /etc/host.conf을 추가적으로 사용한다. 이 파일에는 간단한 2개의 행이 있다.

```
order hosts bind
multi on
```

order명령문은 분석의 순서를 결정한다. 여기서 /etc/hosts는 이름봉사기에 접근되기전에 검색된다. 다음행은 무시한다.

24.6 구성에 대한 검사(ndc, nslookup)

Linux체계상에서 named는 자기의 PID를 /var/run/named.pid안에 보관한다. 이것은 관리자가 그 데몬을 제거 및 재개하기 위하여 재개신호(hangup signal)를 사용할수 있다는것을 의미한다(kill -HUP `cat /var/run/named.pid`). 그렇지만 ndc는 이 데몬을 조작하는 더 좋은 방법을 제공한다. 아래에 인수 start, stop, restart를 받아 들이는 쉘스크립트를 준다.

```
ndc start
ndc stop
ndc restart               ndc start뒤에 ndc stop
```

관리자는 또한 named가 이름봉사기들과 우편교환기들의 이름을 알고 있는가를 시험하기 위하여 nslookup지령을 사용할수 있다. 이 지령은 비대화적으로 FQDN과 함께 리용하여 주콤퓨터의 IP주소들을 얻을수 있다. 여기서 그 지령을 대화적으로 사용하며 이름봉사기를 찾아 보기 위하여 type를 ns로 설정한다.

```
# nslookup
Default Server:  saturn.planets.com
Address:  202.168.0.1
> set type=ns
> planets.com
Server:  saturn.planets.com
Address:  202.168.0.1

planets.com     nameserver = uranus.planets.com
planets.com     nameserver = saturn.planets.com
uranus.planets.com     internet address = 202.168.0.3
saturn.planets.com     internet address = 202.168.0.1
```

type를 mx로 설정하여 named가 우편교환기들을 알고 있는가를 볼수 있다.

```
> set type=mx                 우편교환기레코드들만
> planets.com
planets.com    preference = 20, mail exchanger = saturn.planets.com
planets.com    preference = 10, mail exchanger = jupiter.planets.com
      .............                이름봉사기들을 생략
saturn.planets.com    internet address = 202.168.0.1
jupiter.planets.com    internet address = 202.168.0.2
```

이 구성을 시험하기 위해서는 다음의 지령들을 사용하는것이 더 좋을것이다. 먼저 nslookup은 이름봉사기들과 우편봉사기들을 정확히 식별하였다. 이제 단독의 주콤퓨터이름들과 완전한 FQDN들을 가지고 그 기계들의 일부에 접근하자. 실례로 Web봉사기가 neptune상에서 실행되고 있는가를 보기 위하여 관리자의 열람프로그람상에 www.planets.com을 입력할수 있다. 또한 분석기가 주콤퓨터이름을 FQDN으로 확장하는가를 알기 위하여 ftp uranus를 사용할수도 있다.

참고

이름봉사기가 제대로 실행하고 있는가를 시험하기 위해서는 nslookup을 FQDN과 함께 사용하고(nslookup ftp.planets.com) 그 주콤퓨터의 IP주소가 출력되는가를 보시오.

24.7 우편봉사

전자우편(Email)은 아마 관리자가 인터네트봉사기에 설치하기 위하여 물어 보는 첫 봉사일것이다. 인터네트우편이 다소 복잡하기는 하지만 작은 기관에서의 우편봉사기설정은 그렇게 할 필요가 없다. Rational Planets는 또한 Velvet의 우편을 조종하며 그것은 봉사기설치를 더 복잡하게 한다. 그렇지만 지금까지의 설명에 기초한다면 거의나 정확하게 만들수 있다.

우편전송과 배달수법은 이미 13.6에서 설명되였다. 본질을 돌이켜 보면 인터네트우편은 다음의 세 대행체들이 만든다.

- 우편사용자대행체(Mail User Agent:MUA): 우편을 만들거나 읽는다.
- 우편전송대행체(Mail Transport Agent:MTA): 우편을 보내고 받는다.
- 우편배포대행체(Mail Delivery Agent:MDA): MTA로부터 우편을 받아 사용자들의 우편통에 배달한다.

이 다소 복잡한 설치의 구성을 처리하기전에 우편대행체들사이의 관계를 리해하기 위하여 그림 24-1을 다시 보시오. 오늘날 MTA의 일감은 일반적으로 SMTP(Simple Mail Transfer Protocol:단순우편전송규약)에 의하여 조종된다. sendmail은 SMTP를 실현한 가장 일반적인 프로그람이다. 앞으로 나가면서 sendmail과 그 구성에 대하여 설명한다.

사용자가 회선을 통하여 망에 접속하는 경우 우편은 사용자에게 직접 도달할수 없다. 4번째 대행체가 원격봉사기로부터 우편을 가져 오는 역할을 한다. 이 방식에서 우편을 저장하고 가져 오는 통신규약은 POP(Post office Protocol:우편국규약)와 IMAP(Internet Message Access Protocol:인터네트통보문접근규약)이다. fetchmail은 Linux체계에서 표준POP/IMAP의뢰기이다. POP3봉사기로부터 우편을 받기 위하여 fetchmail을 구성하는 방법을 설명한다.

24.7.1 종합적인 MTA - sendmail

망에서 워크스테이션상의 사용자들은 일반적으로 자기들의 우편을 **집선기**(우편봉사기)에 제출하며 집선기(hub)는 외부세계에 직접 접근한다. 이와 류사하게 워크스테이션은 직접 우편을 받지 않으며 우편은 모두 집선기에 들어 온다. 이 접근법은 우편조작에서 일어 날수 있는 복잡한 혼란으로부터 개별적인 사용자들을 보호한다. 우편통보가 보내여 질수 없다면 송신자에게 통지하기전에 5일동안 계속 시도해 보는것이 집선기의 일감이다.

집선기가 통보문의 경로를 조종할 때 그것은 집선기로부터 출발한것으로 보이도록 송신자주소를 다시 쓴다. 다시 말하여 주소 henry@neptune.planets.com을 henry@jupiter.planets.com으로 고쳐 쓰며 여기서 jupiter는 우편을 조종하기 위한 planets의 집선기이다. 객관은 henry가 우편을 보내기 위해 실제로 사용한 기계를 모른다. 이것은 henry가 다른 주콤퓨터에로 이동할수 있고 이때 같은 전자우편주소를 사용할수 있다는것을 의미한다. 집선기가 전체 령역 planets.com에 관한 우편을 관리한다면 대부분의 조작들은 그 집선기의 이름을 나타내지 않는다(henry@planets.com).

들어 오는 우편의 경로는 집선기를 통하여 조종되므로 사용자는 다음의 3가지 선택항목들을 가진다.

- telnet나 rlogin으로 가입한후에 집선기상의 통보문을 보기: planets의 모든 사용자들이 그 집선기에 등록자리를 가지고 있다면 그렇게 할수 있다.
- 우편이 자기기계에로 발송되도록 요구하기: 이를 위하여서는 자기의 기계가 항상 가동해야 한다.

- 집선기상에서 실행되고 있는 POP/IMAP봉사기로부터 우편을 가져오기: 이것은 Velvet가 Planets의 우편봉사기로부터 우편을 받을수 있는 유일한 방법이다. Planets의 사용자들도 우편이 자기의 기계로 발송되는것을 바라지 않는다면 이 기능을 사용할수 있다.

지금까지 UNIX체계를 위한 가장 공통적인 MTA인 sendmail에 대하여 간단히 소개하였다. MMDF나 smail과 같이 UNIX나 Linux체계상에서 리용할수 있는 다른 MTA들도 있으나 sendmail은 인터네트 우편의 대부분을 관리한다. 이것은 버클리의 에리크 올맨(Eric Allman)에 의하여 작성되였으며 여전히 그에 의해 관리되고 있다. sendmail은 유닉스표준이기는 하지만 아주 복잡하고 이상한 프로그람이다. 사람들은 그의 구성파일로 편집하는것을 어려워 한다.

UUCP처럼 일부 통신규약들과 달리 sendmail은 중간주콤퓨터들을 사용하지 않는다. 그것은 수신측의 또 다른 MTA와 통신하며 확인한 기초우에서 수신측 MTA에 직접 우편을 전달한다. 그 프로그람은 또 다른 sendmail 프로그람일수도 있고 아닐수도 있으나 SMTP를 리해하여야 한다. 발송기능의 설정에 따라 마지막 MTA가 받기전에 우편은 여러개의 MTA들에 의해 조종될수 있다. 만일 통보문이 배포될수 없다면 sendmail은 그 통보문을 대기렬에 넣고 규칙적인 시간간격으로 몇번 시도하여 보고 그 실태에 대하여 사용자에게 알려 준다.

sendmail은 임의의 위치에 있는 한명이상의 사용자들에게 우편을 발송하기 위하여 별명의 개념을 사용한다. 별명들은 파일 /etc/aliases안에 있으며 sendmail은 통보문을 수신할 때 이 파일의 변형(variant)을 읽는다.

24.7.2 sendmail의 시동과 정지

sendmail은 /usr/sbin(Solaris에서는 /usr/lib)안에 있다. pppd와 같이 봉사기구성요소와 의뢰기구성요소 둘 다 가질뿐아니라 단일한 기동으로 동시에 두가지 방식으로 동작 할수 있다. 그것은 포구 25에서 들어 오는 모든 우편을 받아 들이기 위하여 어느 한 스크립트로부터 봉사기데몬으로서 실행한다. 만일 sendmail이 들어 오는 우편만을 수신하고 나가는 우편을 송신하지 않도록 되여 있으면 아래의 방법으로 불러 내야 한다.

```
/usr/sbin/sendmail  -bd
```

-bd선택항목은 sendmail을 데몬프로쎄스로서 실행시키며 MTA와 같이 우편배달은 하지 않는다. 다른 MTA에로 우편을 발송하고 있지 않는 한 그것은 배달프로그람에 우편을 넘겨 준다. UNIX체계들은 /var/spool/mail(SVR4에서는 /var/mail)안의 한개의 본문파일안에서 사용자의 우편을 관리한다. henry에 대한 통보문은 파일 /var/spool/mail/henry에 추가된다.

나가는 우편은 등록부 /var/spool/mqueue에 줄 세워 진다. 만일 sendmail이 나가는 우편만을 송신하도록 되여 있다면 아래의 두 방법중의 하나를 리용하여 의뢰기로서 실행하여야 할것이다.

```
/usr/sbin/sendmail -q              한번 호출
/usr/sbin/sendmail -q 15m          15분마다 대기렬을 시험한다
```

첫번째 경우에 sendmail은 한번의 실행으로써 대기렬을 처리한다. cron봉사를 사용하면 이 지령을 규칙적인 시간간격으로 실행시킬수 있다. 또한 15분마다 대기렬을 감시하고 그것을 내보 내는 2번째 형식을 사용할수도 있다. 데몬방식으로 실행하고 있는 sendmail은 뿌리허가권을 요구하지만 -q선택항목은 모든 사용자들이 사용할수 있다.

sendmail이 -bd선택항목과 함께 데몬으로서 실행된다고 해도 관리자는 그것을 -q선택항목을 가지고

다시 불러 낼수 있다. 많은 경우에 같은 기계가 들어 오고 나가는 우편을 다 조종한다. 인터네트에 항상 련결되여 있는 주콤퓨터상에서 sendmail을 불러 내는 가장 일반적인 방법을 아래에 주었다.

```
/usr/sbin/sendmail -bd -q 15m
```

sendmail의 동작은 /etc/sendmail.cf(Solaris에서는 /etc/mail/sendmail.cf)에 의하여 조종된다. 이 파일의 내용을 변화시킬 때에는 반드시 sendmail을 제거하고 다시 시작해야 한다. 이것은 적절한 스크립트(말하자면 /etc/rc.d/init.d/sendmail)를 restart인수와 함께 실행시키는것에 의하여 수행될수 있다. 만일 관리자의 기계상에 있는 rc스크립트가 start와 stop인수만을 지원한다면 ps지령으로부터 PID를 밝혀 낸 다음 SIGHUP신호를 가지고 sendmail을 제거해 볼수 있다.

Linux체계상에서 sendmail은 /var/run/sendmail.pid의 2개의 분리된 행에 PID와 그것을 실행시키기 위해 사용되는 지령행을 보관한다. 첫행에 보관된 PID를 제거하고 그다음 두번째 행의 내용을 실행시킨다.

```
kill `head -1 /var/run/sendmail.pid`          sendmail을 정지한다
`tail -1 /var/run/sendmail.pid`          sendmail을 시동한다
```

우리는 2번째 행에서 보여 준 방법으로 지령치환을 사용하지 않았지만 이것은 파일안에 매몰된 지령행을 실행시키는 좋은 방법이다. 더 좋은 방법은 그를 위한 별명을 생성하는것이다.

24.7.3 우편대기렬의 현시-mailq

mailq지령을 가지고 우편대기렬을 볼수 있다. 이것은 sendmail지령에 대해 기호련결만 하고 실제로는 sendmail -bp를 실행시키는것과 같다.

```
# mailq
                Mail Queue (1 request)
--Q-ID-- --Size-- ----Q-Time---- ------------Sender/Recipient-------------
QAA00943   478 Fri Dec 10 16:22 <sumit@saturn.planets.com>
                                 <XLDEL/TMH/VIBHA@tmh.satyam.net.in>
```

이것은 일감을 상세하게 보여 주는 첫행에 통보문-ID(QAA00943)와 송신자의 전자우편주소를 현시한다. 수납자는 2번째 행에 있다. 통보문ID는 sendmail이 매 통보문을 위해 생성한 두개의 파일이름속에 매몰된다. /var/spool/mqueue를 렬거하면 이 두 파일을 볼수 있다.

```
# ls /var/spool/mqueue
dfQAA00943
qfQAA00943
```

모든 통보문은 이 등록부안에 있는 두개의 파일로서 격납된다. qf파일은 우편의 머리부를 포함하며 df파일은 내용을 포함한다. 모든 사용자는 이 지령을 실행하여 완충된 모든 통보문의 목록을 볼수 있다. 결국 대기렬로부터 통보문을 제거하는 지령은 없다. 만일 통보문을 삭제해야 한다면 관리자는 대응하는 df과 qf파일에 관하여 rm을 사용해야 한다.

24.8 구성파일 sendmail.cf

sendmail의 내용은 구성파일 /etc/sendmail.cf에서 찾을수 있다. 많은 사람들이 질색하는 이 파일은 거의 천개의 행을 가지고 있으며 거기에는 본적이 없는 부호화된 코드들이 들어 있다. 다행히도 sendmail은 그 내용의 대부분을 리해하지 않고도 거의 모든 소규모 및 중규모체계상에 설정될수 있다. 사용자가 알아야 하는것들은 앞으로 나가면서 설명한다.

드문히 사용자가 sendmail.cf를 서술하여야 한다. 모든 UNIX체계들은 미리 구성된 파일을 가지고 온다. sendmail이 가동하도록 하기 위해서는 거기서 몇개의 행만을 변화시키면 된다. 그렇지만 많은 사용자들이 그렇게 하는것도 피하고 있다. sendmail구성을 간단히 할 필요를 깨닫고 sendmail의 개발자들이 m4마크로처리기를 만들어 냈다. 사용자는 별개의 파일에 있는 몇개의 변수들을 설정하고 m4로 하여금 sendmail.cf를 발생하도록 하면 된다. 여기서 m4는 설명하지 않고 sendmail.cf을 직접 편집하도록 한다.

sendmail.cf는 여러개의 부분으로 나누어 지며 매 부분은 서로 다른 측면의 구성을 취급한다. 이 절에서는 구성파일의 일부 마크로와 클라스들을 분석해 본다. 또한 우편봉사기를 구성할 때 일부 다른 파라메터들도 부닥친다. sendmail은 다른 구성스크립트들과 다르다. 즉 모든 명령문들이 1렬부터 시작해야 한다.

24.8.1 마크로

마크로들은 간단히 변수와 같이 볼수 있으며 파일의 시작부분에 있을수 있다. 모든 마크로정의는 D로 시작하며(sendmail.cf에 의하여 사용되는 문자 M은 우편처리기를 의미한다.) grep "^D" /etc/sendmail.cf를 리용하여 쉽게 그것들의 위치를 찾을수 있다. 중요한 마크로들의 일부를 아래에 보여 준다.

```
Dj$w.planets.COM
DSscarletisp.com
DMplanets.com
DnMAILER-DAEMON
DZ8.9.3
```

문자 D의 바로 뒤에는 한 문자의 마크로이름이 놓이며 그다음에 그의 값이 놓인다(공백의 구분이 없이). 여기서 S는 scarletisp.com으로 설정되며 Z는 8.9.3이라는 값을 가진다. 쉘변수와 같이 모든 마크로변수들의 값은 $와 함께 처리되며 그것들은 sendmail.cf의 여러 토막들에서 참조된다.

많은 sendmail마크로들은 소문자로 시작한다. 이미 그것들중 하나($w)를 보았다. 모든 소문자마크로들은 그 파일안에 정의되지 않고 sendmail안에 내부적으로 정의된다. 머리부분에는 몇개의 마크로들이 있다. 아래에 sendmail이 통보문머리부들의 형식을 결정하는 마크로의 사용법을 보여 주는 몇개의 행이 있다.

```
H?D?Date: $a
H?F?From: $?x$x <$g>$|$g$.
H?x?Full-Name: $x
H?M?Message-Id: <$t.$i@$j>
```

머리부의 매행은 H에 의해 구별되며 그뒤에 물음표(?)로 닫긴 기발과 머리부형태가 놓인다. $a는 체계자료를 RFC822형식으로 보관한다. $g는 송신자의 주소, $x는 완전이름이다.

Message-Id는 대체로 <199908230446.KAA00913@mail.hill.com>과 같은 값을 가지는 긴 문자렬이다. 그것은 수값적으로 표현되는 날자와 시간($t), queue-id($i), 주콤퓨터의 FQDN($j)으로부터 얻어 진다. 두개의 전자우편통보문은 동일한 Message-Id를 가질수 없다.

주해

통보문머리부안에서 From:행은 해석하기가 재미 있게 되여 있다. 여기서 값은 조건적으로 설정된다. 그것은 $x가 설정되였는가 안되였는가에 따라 $x<$g>(RFC822형식)이든가 $g가 된다. $?, $|, $.은 임의의 if구조의 if, else, endif에 약어처럼 작용한다. 해석은 다음과 같다. "If x exists ($?x), the value is $x <$g> else ($|) it is $g endif($.)." 만일 From:행에 완전이름이 나타나지 않는다면 이것은 관리자가 시험해 보아야 하는 행이다.

24.8.2 클라스

sendmail은 클라스를 의미하는 문자 C도 사용한다. 마크로와 같이 이것도 뒤에 한 문자가 놓이며 그다음에는 목록이 놓인다(목록에는 값이 한개일수도 있고 여러개 일수도 있다). 하나이상의 값들의 목록이 놓인다. 여기에 sendmail의 작업을 더 잘 리해할수 있게 하는 몇개의 클라스가 있다.

```
CO @ % !
Cwlocalhost mail.planets.com planets.com velvet.com
CE root
```

첫 클라스 O는 공백으로 구분한 3개의 값을 포함한다. 이 클라스는 전자우편주소의 사용자이름부분에 문자 @, %, !를 사용하는것을 금지한다.

클라스 w는 우편을 받을것으로 예상되는 모든 주콤퓨터들의 FQDN을 포함한다. 이 행에 모든 FQDN들을 지정할수 있지만 너무 많을 때에는 대신 파일클라스(F)를 사용할수 있다. 파일클라스(F)는 파일로부터 목록을 가져 온다는것을 가리키기 위하여 같은 한개의 문자(여기서는 w)와 함께 문자 F를 사용한다. 아래에 100개의 FQDN을 조작해야 할 경우 w를 제대로 사용하는 방법을 주었다.

```
Fw/etc/sendmail.cw
```

Rational Planets에서 모든 우편은 jupiter와 그의 별명 mail이 수신한다. 본명 jupiter를 지정하는 특정한 MX레코드는 있지만 별명 mail을 지정하는것은 없다(MX레코드는 별명으로 된 주콤퓨터는 지정할수 없다). 일단 별명클라스에 mail.planets.com을 추가한다면(Cw나 Fw로) 체계는 user@mail.planets.com 으로 주소가 지정된 우편도 받는다. 뿐만아니라 체계는 user@velvet.com으로 주소화된 우편도 받아 들인다. 이것은 /etc/mailertable안에 Velvet를 위한 항목을 배치하는 방법으로 일부 추가적인 구성을 요구하지만 여기서는 그것을 무시하기로 하자.

24.8.3 선택항목

sendmail은 파일의 위치와 오유통보 및 시간초과의 처리 등을 정의하는 100개이상의 선택항목들을 지원한다. 선택항목들은 문자 O로 정의한다. 판번호 8.7에서부터는 정의가 달라 졌다. 선택항목은 아래의 방법들중 어느 한가지로서 표현된다.

```
OA/etc/aliases
O AliasFile=/etc/aliases
```

첫번째 형식은 공백의 구분이 없이 한개의 문자와 값을 사용한다. 여기서 선택항목 OA는 값 /etc/aliases를 가진다. 둘째 형식은 선택항목이름으로서 완전한 단어 AliasesFile을 사용한다. AliasesFile의 앞에는 공백이 놓이며 뒤에는 같기표(=)가 놓인다. 대소문자구별이 없으며 Aliasfile과 aliasfile은 같은 의미를 가진다. 일부 중요한 선택항목들을 아래에 준다.

```
O HelpFile=/usr/lib/sendmail.hf
O ForwardPath=$z/.forward.$w:$z/.forward
O QueueDirectory=/var/spool/mqueue
O Timeout.queuewarn=4h
O Timeout.queueretrun=5d
```

두번째 행은 sendmail이 .forward파일의 위치를 찾기 위해 따라 가야 할 경로를 지정한다. 그것은 두개의 마크로 $z와 $w를 사용한다. 이 마크로들이 무엇을 표현하는가를 거의나 짐작할수 있다($w는 주콤퓨터이름을, $z는 홈등록부를 표현한다).

관리자는 통보문을 배달할수 없음을 표현하는 우편기데몬(MAILER-DAEMON)으로부터 통보문을 발견해야 한다. 마지막 두 선택항목은 sendmail이 배달을 포기하기전에 얼마나 오래동안 시도하겠는가를 결정한다. 만일 sendmail이 4시간내에 우편을 배달할수 없다면 송신자에게 경고통보를 보내는 한편 시도를 계속 한다. 5일동안 여전히 불가능하다면 포기하고 그 우편을 mqueue등록부로부터 삭제한다.

24.9 별명

sendmail은 임의의 주콤퓨터의 임의의 사용자에게 우편을 보내는 유연하고 다방면적인 특징을 가지고 있다. 이것을 가능하게 하는것이 파일 /etc/aliases에 의하여 표현되는 **별명화**(aliasing)기능이다. 이 파일의 매행은 두 부분으로 나누어 진다. 왼쪽부분은 별명을 가져야 할 사용자이름을 포함하며 오른쪽은 목적지를 포함한다. 더우기 그 파일의 일부 항목들은 몇가지 놀라움을 던져 준다.

```
postmaster: root
mailer-daemon: postmaster
enquiry: henry
charlie: charlie@neptune.planets.com
friends: romeo,juliet@yahoo.com,jack,jill,andrew
navlist: :include: /home/nav/subscriber.lst
hegel: /dev/null
```

여기서 항목들은 이 파일을 사용할수 있는 여러가지 방법을 보여 준다. 모든 사용자이름의 뒤에는 두점(:)이 놓이고 그뒤에 공백이 놓인다. 오른쪽은 별명의 확장이다. 이 확장내용은 여러개의 사용자이름들이 될수도 있고 지어는 파일이름들이 될수도 있다. enquiry로 주소화된 모든 우편은 henry에게 발송된다.

planets.com령역을 위한 주우편봉사기는 jupiter이다. 그러므로 charlie에게로 주소화된 우편은 보통 jupiter에 완충될것이다. 여기서 별명은 그것을 주콤퓨터 neptune에로 다시 보낸다. 이 별명은 몇가지 의미를 가진다. 모든 사용자들이 user@planets.com으로서 집선기에서 수신되였다고 해도 별명은 관리자가 그것들의 일부 또는 전부를 그것들이 실제로 속해 있는 기계에로 다시 보내게 한다.

sendmail은 수신된 통보문의 머리부를 검사할 때 먼저 /etc/aliases와 함께 To:마당안에 있는 사용자이름을 검사한다. 그리고 별명에로 넘어 가서 그 확장내용이 또 다른 별명에로 련결되는가를 계속 검사한다. 여기서 mailer-daemon으로 주소가 지정된 우편은 결국 뿌리의 우편통에 도착한다(왜냐하면 postmaster가 뿌리로 별명되기때문이다).

다른 별명들도 보자. friends로 주소화된 우편은 5명의 사용자들에 의해 수신된다. 그들중 하나는 국부령역안에 있지 않고 인터네트상에 있다. 즉 sendmail은 이것을 허용한다. friends는 한 집단의 사용자들이 단일이름으로 주소화되는것을 허락하는 우편목록(mailing list)을 표현한다. 5명의 사용자들은 /etc/aliases안에 수용하는것이 적당하지만 500명일 때에는 한 파일안에 배치하는것이 더 좋다. :include:예약어를 사용하여 sendmail이 별명 navlist를 확장하기 위하여 파일 subscriber.lst를 찾아 보도록 지시할수 있다.

수납자는 파일일수도 있다. hegel이 자기의 새 주소를 남겨 두지 않고 그 기관을 떠났다면 그에게 오는 모든 우편은 무효로 되여야 한다. 그 통보문을 /dev/null에로 보내는것보다 더 좋은 방법은 없다.

관리자는 사용자가 가입하거나 떠나갈 때 별명을 추가하거나 삭제하기 위하여 이 파일을 수정할수 있다. 그러나 그때 관리자는 newaliases지령을 가지고 그것을 번역해야 한다. 그 지령은 인수없이 실행된다. mailq와 같이 newaliases는 sendmail로 또다시 련결되여 sendmail -bi를 실행한다. 그것은 파일 aliases.db를 2진형식으로 만든다. sendmail은 우편머리부를 검사할 때 aliases가 아니라 aliases.db를 읽는다.

주의

모든 별명파일에 있는 별명 postmaster와 mailer-daemon을 절대로 변화시키지 마시오.

주해

/etc/aliases에서 모든 사용자는 $HOME/.forward안에 목적지전자우편주소의 이름을 배치하는것으로 자기자신의 발송을 할수 있다. 림시발송은 관리자에 대한 참조가 없이 .forward에 의하여 수행된다. 그렇지만 사용자가 조직에서 나가거나 옮길 때 그 사용자의 등록자리가 삭제될수 있고 그의 홈등록부가 제거될수 있기때문에 /etc/aliases를 선택하는것이 더 좋다. 그때는 .forward도 사라진다.

24.10 중심국을 위한 우편봉사기의 설치

planets.com망에서 령역의 자료기지안에 있는 MX레코드는 jupiter와 saturn에게 전체 령역에 대한 우편을 관리할 책임을 할당한다. jupiter는 saturn보다 더 높은 우선권을 가진다(jupiter는 10, saturn은 20). 모든 주콤퓨터는 자기들의 우편을 jupiter에로 발송하므로 그것들은 sendmail을 데몬방식(-bd)으로 실행시킬 필요가 없고 대기렬방식(-q)으로만 실행시키면 된다. jupiter와 saturn만이 sendmail을 데몬방식으로 실행시켜야 한다.

이 설정에서 jupiter와 saturn을 제외한 모든 주콤퓨터들은 sendmail.cf구성으로 간단히 할수 있다. 실례로 neptune안에 있는 그 파일은 우편이《지능화된》중계주콤퓨터 jupiter에게로 발송된다는것을 나타내는 행을 포함한다. 이것은 neptune의 sendmail.cf에 있는 S마크로에 의해 이루어 진다.

```
DSjupiter.planets.com                지능화된 중계를 설정
```

왜 중계봉사기가 필요한가? 인터네트에 접속되지 않았거나 우편을 배달하는 능력이 부족한 주콤퓨터들은 그 어느쪽도 제한을 가지지 않는 주콤퓨터가 필요하다. 우편을 보내는것은 때때로 곤난한 문제가

될수 있다. 수신기계가 정지될수도 있고 선로파손의 리유로 그 주콤퓨터까지의 경로가 없을수도 있다. 믿음직한 우편봉사기가 상대측의 MTA에 접근하기 위하여 적당한 시간동안 시도해 본후 포기해야 한다.

전자우편주소에 주콤퓨터이름을 나타내지 않는것이 일반적인 기관들의 방책이다. 그 경우에 jupiter는 jupiter.planets.com이 아니라 planets.com으로 변장(masquerade)되여야 한다. 관리자는 아래와 같은 방법으로 jupiter상에 M마크로를 설정해야 한다.

```
DMplanets.com
```
planets.com으로 변장

이것은 우편을 내보내는 jupiter상의 사용자 henry가 henry@jupiter.planets.com대신에 henry@planets.com으로 고쳐 진 송신자의 이름을 가진다는것을 의미한다. 이것은 jupiter의 사용자들에게는 좋지만 Planets는 neptune, uranus, saturn 등을 포함하는 전체 planets.com령역의 송신자들을 위한 통일적인 구조를 가지려 할수도 있다. 그러한 경우에 이러한 **변장**(masquerading)은 jupiter에로 우편을 발송하는 모든 주콤퓨터들에 대하여 수행되여야 한다. 이번에는 클라스 M을 사용해 보자.

```
CM neptune.planets.com saturn.planets.com uranus.planets.com
```

주해

M클라스(CM)를 가지고 전체 령역을 변장할수 있게 하기 위해서는 전체 기관의 사용자이름들이 유일적인가를 확인해야 한다. 이것은 사용자등록자리들을 조직하는 좋은 방법이기도 하다. 사용자이름들이 령역전반에서 유일적이지 않다면 CM을 사용할수 없다.

이제는 Planets의 우편봉사기가 자체로 사용할수 있게 준비되였다. 그러나 Planets는 Velvet의 우편봉사도 조종해야 하며 sendmail이 그것을 알아야 한다. 이것은 w클라스의 일감이다.

```
Cwlocalhost planets.com velvet.com
```

이것으로 우편구성이 끝난다. 그렇지만 아직 수신된 우편의 처리에 대하여 결정해야 한다.

planets.com망에 대한 간단한 우편구성에서 jupiter는 인터네트에로의 영구적인 접속을 가지고 있으며 /var/spool/mail안에 모든 사용자에 대한 우편을 격납한다. 사용자들이 그 우편에 어떻게 접근하는가? 만일 사용자들이 telnet를 가지고 jupiter에 가입한다면 그것들은 elm이나 pine과 같은 MUA를 사용하여 자기들의 우편을 볼수 있다. 그것들이 Netscape(또는 자기들의 기계에 원래부터 있는 MUA들)를 사용하고 있다면 그것들은 자기의 기계상에 우편을 내려 받아야 한다. 여러가지 해결책이 있으며 Planets는 여러가지 워크스테이션에 대하여 여러가지 해결책을 가지도록 선택할수 있다.

- 주콤퓨터가 항상 가동중이며 jupiter에 접속되여 있다면 그 주콤퓨터에 속하는 사용자들에 대한 우편을 발송하기 위하여 /etc/aliases를 사용할수 있다. 그때 jupiter는 이러한 모든 사용자들을 /etc/passwd안에 가지고 있어야 한다.
- 모든 주콤퓨터가 jupiter에 상시적으로 접속되여 있다면 모든 우편이 그것들에 발송되든지 아니면 NFS를 사용하여 /var/spool/mail파일체계를 사용자들이 원격으로 설치하는것이 허락될수도 있다.
- 주콤퓨터가 jupiter에 때때로 접속된다면(대체로 전화선을 통하여) jupiter는 POP 또는 IMAP봉사기로도 동작해야 한다. 이 주콤퓨터상의 사용자들은 POP나 IMAP통신규약을 리용하여 자기들의 우편을 가져 올수 있으며 주콤퓨터 jupiter상에 등록자리를 가지고 있어야 한다.

/etc/aliases를 어떻게 사용하는 방법과 NFS파일체계를 설치하는 방법에 대하여서는 설명되였다. 이제 그림 24-1을 참고하면 주콤퓨터 mercury가 전화선을 통하여 jupiter에 접속하고 있다는것을 볼수 있다. 이것은 3번째 선택항목을 표시하며 이 선택항목을 사용하고 있는 주콤퓨터들은 별개의 우편구성을 요구한다.

24.11 비직결사용을 위한 통신규약 POP와 IMAP

수신우편봉사기는 인터네트에 상시적으로 접속되여 있어야 하지만 워크스테이션들은 이 봉사기에 항상 접속되여 있을 필요가 없다. 사용자들은 종종 밤에는 자기의 기계를 끄기때문에 /etc/aliases를 통하여 우편을 발송할 모든 기회를 배제한다. 개인들도 규칙적인 시간간격으로 전화선을 통하여 자기의 ISP 우편봉사기에 접속한다. 이 경우들은 보내주는것보다 오히려 꺼내오는것을 요구한다. POP와 IMAP가 거기에 사용되는 통신규약이다.

Planets망에서 만일 사용자들이 원격으로 /var/spool/mail을 설치하는것이 허용되지 않는다면 jupiter는 우편을 POP나 IMAP봉사기에 격납해야 한다. 여기서 거의 모든 인터네트봉사기들에서 표준으로 되는 POP3을 고찰한다. IMAP는 아직 우수한 표준으로 채택되지 않고 있다. POP3이 가능하게 될 때 우편파일위치들은 그것들을 이 규약을 가지고 있는 원격기계로부터 가져 올수 있다는것을 제외하고는 여전히 같다.

POP3은 /etc/inetd.conf로부터 inetd에 의하여 시작된다. 그것이 실행되기 위해서는 jupiter가 파일안에 아래와 류사한 행을 포함해야 한다.

```
POP3    stream  tcp   nowait  root    /usr/sbin/tcpd   /usr/sbin/popper -s
POP-3   stream  tcp   nowait  root    /usr/sbin/tcpd   ipop3d
```

Red Hat와 SuSELinux의 이러한 설정에서는 외피프로그람 tcpd가 popper나 ipop3d프로그람을 실행한다. POP3은 포구번호 110을 사용하며 이것이 /etc/services안의 해당한 항목안에 반영되여야 한다.

```
POP3         110/tcp    # POP version 3
POP-3        110/tcp    # POP version 3
```

jupiter의 /etc/inetd.conf안에서 그 행이 설명문화되지 않았는가를 확인하여야 한다. 그러면 jupiter의 POP봉사기로부터 우편을 내려 받을 필요가 있는 주콤퓨터들에 대한 우편구성을 보자. 이것은 직접적으로 접속되여 있으면서 여전히 POP를 사용하는것을 더 좋아 하는 주콤퓨터들은 물론 회선을 통하여 접속하는 주콤퓨터(mercury와 같은)들도 포함한다.

POP/IMAP의뢰기 fetchmail

Linux는 POP나 IMAP봉사기로부터 우편을 가져 오기 위하여 fetchmail이라는 재치 있는 프로그람을 사용한다. 이 의뢰기프로그람은 사용자이름, 통신규약, 우편봉사기의 FQDN을 인수로 요구한다. 아래에 mercury상의 사용자 romeo가 어떻게 jupiter에 접속하여 자기의 우편을 받아 오는가를 보여 준다.

```
$ fetchmail -u romeo -p POP3 jupiter
Enter password for romeo@jupiter: *******
2 messages for romeo at jupiter (121400 octets).
reading message 1 of 2 (2354 octets) .. flushed
reading message 2 of 2 (9786 octets) ......... flushed
```

fetchmail은 통과암호를 문의하는데 그것은 화면상에 표시되지 않는다. 그다음은 POP봉사기로부터 우편을 가져 오는 처리를 한다. 통보문들이 봉사기로부터 삭제된다(fetchmail의 기본동작). 다만 더 새로운 통보문들이 이 호출에 의하여 꺼내진다.

fetchmail은 mercury(또는 moon)와 같은 주콤퓨터상에서 실행되는 SMTP와 우편봉사기 jupiter나 ISP의 봉사기(scarletisp와 같은)상에서 실행되는 SMTP사이에 접속되는 련결로서 봉사한다. 그렇지만 총체적인 대면부는 아주 통일적이며 단계들은 무시되지 않는다. 여기서 오유를 범하지 말아야 한다. 즉 mercury도 데몬방식으로 sendmail을 실행하여야 하며 fetchmail은 얻은 우편을 sendmail에 넘겨야 한다. sendmail은 그다음 procmail과 같은 배포프로그람에게 우편을 보낸다. 이것은 mercury가 jupiter에 직접 접속된 경우였다. Velvet의 우편을 관리하는 주콤퓨터 moon에 대하여서도 같은 고찰방법이 적용된다.

관리자가 자기의 망에서 실험하고 있을 때 아마 자기가 무엇을 하고 있는가를 확인할 때까지 이 통보문들이 그 봉사기상에 보관되기를 바랄것이다(-k). 기본적으로 fetchmail은 새로운 통보문만을 받지만 그것도 재정의하여 모든 통보문을 얻도록 요구할수 있다(-a). 그때 그것들을 더 후에 삭제(flush)하도록 할수 있다(-F).

```
fetchmail -u romeo -p POP3 -k jupiter.planets.com          모든 통보문들을 가지고 있다
fetchmail -u romeo -p POP3 -a jupiter.planets.com          모든 통보문들을 얻는다
fetchmail -u velvet -p POP3 -F mail.scarletisp.com         모든 통보문들을 삭제한다
```

검색된 우편은 이제는 본문파일 /var/spool/mail/username안에서 리용할수 있다. romeo는 우편을 보기 위해 elm이나 pine과 같은 문자기반의 의뢰기를 사용할수 있다. 좀 별개의 문제이지만 Netscape에서 $HOME/nsmail에 들어 있는 모든 우편들을 요구할수 있다. 이때 기호련결을 사용하여 임의의 우편파일을 Netscape에서 볼수 있는 임의의 장소에 위치하도록 할수 있다. 더우기 Netscape는 fetchmail - sendmail경로를 거치지 않고 jupiter로부터 우편을 직접 가져 올수 있다.

관리자는 아마 fetchmail을 데몬방식으로 실행시키고 스크립트를 통하여 통과암호를 보내려고 할것이다. fetchmail은 모든 fetchmail선택항목들이 보관될수 있는 .fetchmailrc를 사용한다. 여기에 봉사기상에 통보문들을 남겨 두는 POP3을 리용한 간단한 항목을 보여 준다.

```
poll jupiter.planets.com proto POP3
user romeo with password d276y45t
keep                -k선택항목과 같다
```

poll문은 우편봉사기를 인수로서 사용하며 proto는 사용되는 통신규약의 형태를 지정한다. 여기서는 통과암호가 명확한 본문으로 보관되기때문에 그 파일은 아무에게나 해석되지 않도록 해야 한다. 그 파일의 허가권이 600으로 설정되지 않으면 fetchmail이 실행하지 않는 경우도 있다.

주해

mercury가 POP봉사를 실행하고 있다면 jupiter상의 POP봉사기로부터 가져 온 우편은 mercury의 POP봉사기에 격납된다. 그다음 또 다른 주콤퓨터가 mercury로부터 우편을 가져 올수 있다.

24.12 위성국을 위한 우편체계의 설치

이번에는 Velvet의 주콤퓨터 moon상에 우편을 주고 받기 위한 우편봉사기를 구성해야 한다. Velvet는 번호호출PPP등록자리를 통해서만 Planets의 주콤퓨터 jupiter와 ISP의 주콤퓨터에 접근할수 있다. 이번에는 종전과 달리 우편이 아래와 같은 두개의 원천으로부터 내리적재되여야 한다고 가정한다.

- Velvet가 다중배포우편통(multidrop mailbox; 여러 사용자들에 의하여 공유되는 우편통)을 가지고 있는 jupiter상의 POP봉사기로부터.

  Velvet는 또한 몇명의 사용자들에게 user@velvet.com형식의 전자우편주소를 사용하여 봉사하여야 한다.
- ISP의 주콤퓨터 mail.scarletisp.com상에서 실행하고 있는 POP봉사기로부터.

  거기서 Velvet는 전자우편주소 velvet@scarletisp.com으로 회선접속PPP등록자리를 가지고 있다.

Velvet가 우편을 보내려면 ISP가 재치 있는 중계주콤퓨터로서 동작해야 한다. 따라서 moon상의 sendmail.cf는 moon으로부터 나가는 모든 우편이 PPP를 사용하는 이 중계주콤퓨터에로 발송되도록 구성되여야 한다. 변장(Masquerading)은 또한 모든 주소가 user@velvet.com형식으로 되여 보이도록 moon상에서 실현되여야 한다.

```
Dsmail.scarletisp.com               중계주콤퓨터
DMvelvet.com               변장기능
```

planets.com령역의 주콤퓨터 jupiter와 달리 moon은 인터네트에 이따금씩만 접속한다(ISP를 통하여). 따라서 중계주콤퓨터가 필요하다. 이때 moon상에서 실행되고 있는 sendmail에게 우편전송값이 비싸다는것과 우편을 수신하자마자 곧 배포하려고 시도하지 말아야 한다는것을 알려 주어야 한다. 이것은 관리자가 이 선택항목을 변화시킬것을 요구한다.

```
O HoldExpensive=True
```

관리자는 우편기정의에서도 일부를 변화시켜야 한다. 우편기들은 M으로 정의되며 sendmail.cf에는 4가지 형의 SMTP우편기들이 정의되여 있다.

```
Msmtp,     p=[IPC], F=mDFMuXe, S=11/31, R=21, E=\r\n, L=990,
           T=DNS/RFC822/SMTP,
           A=IPC $h
Mesmtp,    P=[IPC], F=mDFMuXae, S=11/31, R=21, E=\r\n, L=990,
           T=DNS/RFC822/SMTP,
           A=IPC $h
Msmtp8,    P=[IPC], F=mDFMuX8e, S=11/31, R=21, E=\r\n, L=990,
           T=DNS/RFC822/SMTP,
           A=IPC $h
Mrelay,    P=[IPC], F=mDFMuXa8e, S=11/31, R=61, E=\r\n, L=2040,
           T=DNS/RFC822/SMTP,
           A=IPC $h
```

우편기 smtp는 일반적인 SMTP우편을 관리한다. esmtp는 확장된 SMTP를 사용한다. 부호화되지 않은(unencoded) 8비트우편은 이제는 smtp8에 의하여 관리된다. relay우편기는 중계주콤퓨터를 통하여 모든 우편의 경로를 조종한다. 매 우편기정의는 또한 여러개의 인수를 가진다. P는 우편기프로그람의 경로(path)로서 [IPC]를 보여 준다. 즉 이것은 sendmail프로그람을 표현한다. S와 R는 sendmail이 송신자와 수신자의 머리부를 고쳐 쓰기 위하여 사용하는 규칙을 보여 준다.

다른 인수들은 통보문의 개별적인 행들이 \r\n으로 끝난다는것(E)과 행들이 990문자를 초과할수 없

다는것(L)을 보여 준다. 주콤퓨터이름들은 DNS로 해석되고 전자우편주소들은 RFC822형을 가지며 sendmail지령자체가 우편기를 실행시키기 위하여 사용된다(SMTP).

F는 우편기를 위하여 적용될수 있는 기발들을 표현한다. 이 기발들의 대부분은 $a, $b 등과 같이 마크로와 공통적이다. 기정설정들은 그대로 두고 F의 마지막에 e를 추가하였다. 이것은 우편기를 비싸게 만들며 그것은 우편을 즉시에 배포하기 위한 접속을 시도하려 하지 않고 오히려 sendmail -q가 실행되기를 기다린다.

주해

판번호 8.9가 시작되면서 sendmail은 기본적으로 중계를 하지 않으며 통보문 《Relay access denied》를 현시한다. 만일 관리자의 우편봉사기를 통하여 다른 망들이 자기의 우편을 보낼수 있도록 이 기능을 가능하게 하려면 그 령역이나 망주소를 /etc/mail/relay-domains안에 입력하고 sendmail을 재시동하여야 한다.

우편배달은 ISP에로 회선접속을 하고 량측에서 PPP봉사를 시작한 다음 sendmail -q지령을 실행시키도록 스크립트를 서술하는것에 의하여 수행된다. 관리자는 회선접속을 위하여 dip, chat(또는 새로운 도구로서 wvdial)를 사용할수 있으며 관리자의 crontab파일로부터 그 스크립트를 실행시킬수 있다. pppd가 접속시에 /etc/ppp/ip-up을 실행시키므로 sendmail문을 거기에 배치할수 있다. sendmail이 항상 데몬방식으로 실행하고 있다는데 주의를 돌려야 한다(sendmail -bd).

여기서 Velvet가 jupiter상에 가지고 있는 다중배포우편통에 대하여 한마디 하기로 하자. sendmail은 전자우편이 하나의 사용자등록자리로 발송되도록 하나의 령역내의 여러 사용자들에게 주소화되는것을 허용한다. fetchmail은 이 등록자리로부터 우편을 얻어야 한다. 우리가 취급해야 할 우편통이 2개이므로 (하나는 mail.velvet.com, 다른 하나는 mail.scarletisp.com) moon상의 .fetchmailrc는 두 묶음의 명령문을 가진다.

```
poll mail.scarletisp.com proto POP3
user velvet with password bnet6797
poll jupiter.planets.com proto POP3              혹은 mail.velvet.com
no dns aka velvet.com               이 명령문이 필요하다
user pop1244238 password hf789bfd is * here
```

첫 묶음은 ISP의 우편봉사기로부터 Velvet의 우편을 내려 받는다. 두번째 묶음은 Planets가 Velvet에게 할당한 우편통사용자이름 pop1244238을 사용한다. 이 우편통은 여러 사용자들의 우편을 포함하므로 fetchmail은 모든 우편머리부를 검사하여 우편이 여러 사용자들의 우편통에 배포되는가를 확인해야 한다. *가 이 배포를 실현한다. 관리자가 여기서 사용자이름을 가지고 있다면 그때는 모든 우편이 그 사용자의 우편통에 떨어 진다.

moon상에 POP봉사기가 실행되고 있다면 Windows기계로부터의 개별적인 사용자들은 Netscape Messenger와 Outlook Express에 장비된 POP의뢰기기능을 리용하여 자기들의 우편을 내리적재할수 있다.

주해

Netscape Messenger는 POP의뢰기이기도 하며 POP봉사기로부터 우편을 가져 올수 있다. 그렇지만 fetchmail은 한번의 기동으로 여러 봉사기로부터 우편을 받을수 있기때문에 우월하다. Netscape Communicator 6도 이 특징을 가진다. 그러나 fetchmail은 분기우편통으로부터 우편을 받을수도 있다(Netscape는 그렇게 할수 없다).

24.13 Web봉사

Web는 인터네트상에서 가장 보편적인 봉사이다. Web봉사기는 일반적으로 **httpd봉사기**로 알려 져 있다(포구 80에서 실행되는 데몬뒤에 붙여 진 이름). Web상에서 사용되는 몇가지 형태의 httpd봉사기들이 있으며 그들중 대부분은 NCSA의 원본설계에 기초하고 있다. 초기봉사기에는 결함이 여러개 있어 그것들을 수정하여야 한다. 어떤 팀의 사람들이 체계적인 방법으로 이 과제를 수행하여 《쪽무이봉사기(a patchy server)》를 만들어 내였다. 그들은 그것을 Apache(아파치)라고 불렀는데 Apache는 오늘날 Web상의 모든 봉사기들의 절반이상을 차지하고 있다. Apache는 모든 Linux체계에서 리용할수 있는 표준봉사기이다. 판번호는 모든 UNIX변종들에서 발전되여 왔다.

Web봉사기는 의뢰기(열람프로그람)로부터의 요구를 조작하기 위하여 HTTP통신규약을 사용한다. 일반적으로 그 요구는 HTML문서와 그와 관계된 도형자료들을 가져 오기 위한것이다. 외부CGI프로그람의 도움말을 리용하여 Web봉사기는 동적HTML출력도 봉사할수 있다. 한 기계상의 Web봉사기가 여러개의 Web싸이트로서도 봉사할수 있다. 때문에 보안이 매우 중요한 개념이며 Apache는 등록부와 주콤퓨터, 사용자준위에 보안을 적용하는 종합적인 체계를 가지고 있다. 이제 Web봉사기를 자체로 설정하기 위하여 이러한 기능의 대부분을 설명한다.

쪽무이봉사기 - Apache

Linux체계상에서 Apache는 /usr/sbin안에 httpd로서 위치한다. ftp, telnet, pop-3와 같이 그것은 inetd에 의하여 시작될수 있으나 보통 어느 한 rc스크립트로부터 체계기동시에 단독방식으로 기동된다. httpd는 뿌리로서 실행하지만 의뢰기요구를 봉사하기 위하여 개별적인 httpd프로쎄스들을 생성한다. ps axf지령의 출력으로부터 그것을 알수 있다.

```
270  ?  S    0:03 /usr/sbin/httpd ?f /etc/httpd/httpd.conf ?D SSL
280  ?  SW   0:00  \_ (httpd)
281  ?  SW   0:00  \_ (httpd)
282  ?  SW   0:00  \_ (httpd)
283  ?  SW   0:00  \_ (httpd)
284  ?  SW   0:00  \_ (httpd)
```

여기에 PPID(rc스크립트로부터 실행되는 초기프로쎄스의 PID)로서 270을 가지는 5개의 추가적인 프로쎄스가 있다. named나 sendmail과 같이 Apache는 기본httpd프로쎄스의 PID를 파일 /var/run/httpd.pid안에 보관한다. 이것은 kill `cat /var/run/httpd/pid`를 가지고 그 프로쎄스를 제거할수 있다는것을 의미한다. 또한 구성파일을 변화시킨후에는 재개신호(hangup signal)를 사용하여 그 프로쎄스를 재시동할수 있다(kill -HUP `cat /var/run/httpd/pid`).

Apache의 동작은 3개의 구성파일 httpd.conf, srm.conf, access.conf(NCSA봉사기로부터 내려오는 파일)에 의하여 조종된다. 판번호 1.3이상부터 Apache는 한개의 구성파일 httpd.conf를 리용한다. 그 파일은 Red Hat에서는 /etc/httpd/conf안에, SuSE에서는 /etc/httpd안에 위치한다.

기정값에 의하여 Apache는 관리자의 Linux체계상에 완전히 구성되며 체계재설치시에 다른 설정을 하지 않아도 실행될수 있다. 관리자가 구성파일에서 변화시켜야 할것은 많지 않다. 대규모의 설치들을 하여 한 기계가 여러개의 령역을 위한 Web봉사기로서 동작할수 있다(아파치에 의하여 할수 있는 기능).

그러한 경우에 매 가상주콤퓨터와 그의 등록부접근권한의 조종을 위하여 이 파일들에 각각의 항목을 만들어야 한다. 지어는 효과적인 조작을 위하여 어떤 파라메터들을 조정하여야 한다.

24.14 구성파일 httpd.conf

여기서는 파라메터들(Apache는 그것들을 지령이라고 부른다.)을 기능별로 나누어 설명한다.

24.14.1 기초설정

Apache는 ServerRoot지령을 사용하여 그 구성파일이 위치하고 있는 등록부를 설정한다. 그 파일이 거기에 존재하는가를 확인하여야 한다.

```
ServerRoot /etc/httpd
```

Apache는 처음 httpd.conf를 찾고 그다음 srm.conf와 access.conf를 찾는다. 관리자가 모든 내용을 httpd.conf파일안으로 옮긴다고 해도 Apache는 이 파일들을 무시하라고 지령을 주지 않는한 계속 찾을것이다.

```
ResourceConfig /dev/null
AccessConfig /dev/hall
```

또한 httpd의 PID파일과 그의 접근 및 오유기록파일들의 기억위치와 관련한 명령들도 있다.

```
PidFile /var/run/httpd/pid
ErrorLog /var/log/httpd/error_log
Customlog /var/log/httpd/access_log common
```

Apache는 포구 80을 리용한다. 1024이하의 모든 포구번호들은 특권을 받은 포구로 간주된다. 그렇지만 뿌리권한을 가지고 실행하는것은 기본httpd프로쎄스(master httpd process)이다. 실제적으로 항상 작업하고 있는 새끼httpd프로쎄스(child httpd process)들은 보통의 사용자권한을 가지고 실행한다. 안전대책으로서 httpd가 사용하는 UID와 GUID가 임의의 다른 사용자에 의하여 사용되지 않도록 확인하는것이 중요하다. Red Hat는 그 두가지를 위하여 nobody를 사용한다.

```
User nobody          /etc/passwd에 있는 nobody
Group nobody          /etc/group에 있는 nobody
```

Web봉사기관리자(Web봉사기를 유지할 책임을 지닌 사람)의 주소는 기정값에 의해 root@localhost로 설정된다. 이것은 사용자들이 봉사기상에서 문제점에 부딪칠 때 자주 볼수 있는 주소이다. Web봉사기관리자가 뿌리등록자리를 사용하지 않을수도 있고 그 주콤퓨터가 자체의 FQDN을 가질수도 있기때문에 관리자는 ServerAdmin과 ServerName지령을 완전한 의미를 가진 값으로 설정해야 한다.

```
ServerName www.planets.com
ServerAdmin webmaster@planets.com
```

ServerName은 봉사기가 의뢰기로 귀환하는 이름이다. 실례로 Web봉사기가 주콤퓨터 neptune상에서 실행하고 있다고 해도 관리자는 사용자들이 URL창우에 www.planets.com이라는 이름을 보도록 하고 싶을것이다. 관리자는 그 주콤퓨터의 본명과 차이나는 이름을 귀환하도록 ServerName을 설정하기전

에 Web봉사기를 접근하기 위하여 사용된 DNS의 별명기능을 확인해야 한다(CNAME레코드로서). 지역파일은 별명으로서 www를 렬거하며 따라서 ServerName을 우에서처럼 합리적으로 설정할수 있다.

24.14.2 모듈적재

Apache는 여러개의 모듈들로 구성되며 모듈들의 대부분은 쏘프트웨어구성처리시에 조립된다. 이 모듈들은 Web봉사기의 기초적인 기능을 제공한다. 지령들의 대부분이 모듈특성이므로 개개의 모듈들이 리용가능하지 않는 한 부분적인 지령들이 작업하리라고 기대할수 없다. 판번호 1.3에서부터 Apache는 실행시간에 모듈들을 동적으로 적재할수 있으며 따라서 관리자는 더 작은 실행프로그람으로 시동하여 요구되는 모듈들만을 추가할수 있다.

이 모듈들을 유효하게 하는 두개의 지령이 있다. LoadModule지령은 실행가능한 모듈 즉 확장자 .so를 가진 목적파일을 적재하며 거기에는 아래와 같은 행들이 들어 있다.

```
LoadModule mime_module      /usr/lib/apache/mod_mime.so
LoadModule cgi_module       /usr/lib/apache/mod_cgi.so
LoadModule proxy_module     /usr/lib/apache/libproxy.so
LoadModule auth_module      /usr/lib/apache/mod_auth.so
```

매 모듈에 관하여 대응하는 AddModule지령도 배치될수 있다. 이 지령은 실행가능한 모듈을 만들기 위하여 사용된 원천프로그람을 적재한다. 원천프로그람은 항상 .c로 끝난다.

```
AddModule mod_mime.c
AddModule mod_cgi.c
AddModule mod_proxy.c
AddModule mod_auth.c
```

모든 모듈을 적재하지 않아도 된다. 즉 불필요하게 봉사실행률을 떨구지 않게 할수 있다. 요구하는 모듈들만을 적재하면 된다. 실례로 관리자의 싸이트가 CGI를 사용하지 않는다면 cgi_module을 적재하지 않아도 된다.

24.14.3 자원위치

이 구성파일에서 Apache는 봉사기의 Web페지 즉 HTML문서를 보관할 별도의 등록부를 요구한다. 이것을 **문서뿌리등록부**(document root directroy)라고 한다. HTML파일들을 위한 등록부는 DocumentRoot로 설정된다.

```
DocumentRoot /home/httpd/html
```

이것은 관리자가 URL http://www.planets.com/survey.html로서 Web페지에 접근할 때 실제로는 그 파일체계상의 파일 /home/httpd/html/survey.html에 접근하고 있다는것을 의미한다. Apache의 보안기능은 이 계층의 웃등록부로 이동하거나(말하자면 /usr/sbin등록부로) 그 안의 파일에 접근하는것을 허용하지 않는다(닉명ftp도 역시 그것을 허용하지 않는다).

CGI프로그람은 Web상의 대부분의 새치기들을 받아 들이며 그것들은 DocumentRoot가 아닌 어떤 등록부에 보관되여야 한다. 이 등록부는 항상 cgi-bin이며 ScriptAlias로 설정된다.

```
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
```

ScriptAlias는 프로그람들만을 포함하는 등록부에 대한 별명을 설정한다. 이것은 다른 등록부들의 별명을 정의하는데는 사용될수 없다. 이이콘(쪼각그림)들도 자기의 등록부를 가질수 있으나 이때의 별명은 Alias로 수행되여야 한다.

```
Alias /icons/ /home/httpd/icons
```

Alias문은 임의의 등록부의 경로를 설정하는데 유용하다(CGI프로그람을 포함하는 등록부는 제외). 하나의 주콤퓨터가 여러 사용자들에게 봉사할수 있으며 이 사용자들은 자기들의 페지를 위한 개개의 등록부가 있어야 한다. 이것은 UserDir에 의하여 결정되며 기정값은 public_html로 설정된다.

```
UserDir public_html
```

이 등록부는 그 사용자의 홈등록부안에 만들어 진다. 그러므로 Apache가 www.planets.com/´juliet를 위한 URL요구를 만난다면 등록부 /home/juliet/public_html로부터 기정 Web페지를 제공해야 한다(홈등록부들이 /home안에 배치된다고 가정하고). Apache는 또한 UserDir등록부에 대한 선택적인 접근을 허락한다.

```
UserDir disable                  모든 사용자들은 불가능
UserDir enable henry romeo juliet                이 세개의 사용자들을 제외
```

Apache는 자기가 부과한 보안규정을 사용자들이 재정의할수 있게 한다. 그것은 사용자정의된 규정을 포함하는 파일의 이름을 지정한다.

```
AccessFileName .htaccess
```

이것은 모든 사용자들이 호출권을 제각기 설정하려는 모든 등록부에 .htaccess파일을 배치할수 있다는것을 의미한다. 이 파일은 구성파일안에 사용된 많은 지령들을 포함할수 있다. 그러나 그것들은 그 등록부와 그의 보조등록부들에만 적용된다.

24.14.4 파일형

기정Web페지(default Web page)란 무엇인가? 이 등록부이름만으로 끝났을 때 Apache는 지령 DirectoryIndex를 사용하며 그 등록부안의 한개이상의 파일들을 찾아 본다. 기정값에 의해 이것은 index.html로 설정되지만 여러 사용자를 포함하는 싸이트들에서는 총체적인 기정값묶음을 찾는것이 보통이다.

```
DirectoryIndex default.htm index.htm index.html index.cgi welcome,html
```

Windows 3.1사용자들을 수용하기 위해서 그 설정은 3문자확장자도 받아 들인다. 그 등록부에서 아무 파일도 발견되지 않는다면 Apache는 그 등록부의 목록을 현시한다. 현시를 조종하고 표면적인 촉감을 추가할수 있게 하는 훌륭한 선택항목들이 있지만 아래의 설정을 해주는것이 가장 좋다는 점을 제외하고는 그것들을 취급하지 않는다.

```
FancyIndexing on
```

Apache는 새로운 부호화된 파일형을 정의하며 자체로 그것들을 조종한다. 실례로 compress와 gzip로 압축된 파일들은 uncompress나 gunzip프로그람들을 불러 냄이 없이 열람기에 의하여 풀어 질수 있다.

```
AddEncoding x-compress Z
AddEncoding x-gzip gz
```

Mosaic와 Netscape는 둘 다 이 압축해제를 자동적으로 수행할수 있다. 그러므로 열람기로부터 얻어진 등록부목록에서 .gz파일을 찰칵하면 즉시에 압축이 해제된 파일을 볼수 있다.

설명된 내용은 한개 싸이트의 Web페지를 봉사하는 주콤퓨터에는 잘 적용된다. 대규모설치에서는 자주 가상주콤퓨터관리(virtual Hosting)의 개념을 리용하여 여러개의 싸이트들을 봉사한다. 그러한 경우에 매 싸이트는 여전히 이 모든 등록부들을 위한 서로 다른 위치(대체로 자기자체의 홈등록부아래에)를 가진다. 가상주콤퓨터의 구성에 대하여 후에 설명한다.

24.14.5 봉사기의 조정

화려한 내용은 그 싸이트상에 많은 요청을 끌어 들인다. 하루에도 많은 싸이트들에서 수십만개의 요청을 받는것이 례사로운 일이다. 기정Apache구성은 매 의뢰기요청의 봉사에 각각의 httpd새끼데몬들을 생성해야 하므로 이것을 할수 없다. 그러한 프로쎄스를 생성하기 위하여 의뢰기요청을 기다리기보다 오히려 개시시간을 줄이기 위하여 몇개의 프로쎄스들이 준비되여 있게 한다. 이것은 아래의 설정에 의하여 조종된다.

```
MinSpareServers 5
MaxSpareServers 20
StartServers 8
```

임의의 시간에 5개의 프로쎄스들이 요청을 봉사하기 위하여 기다리고 있다. 작업중 싸이트에서는 유지될 필요가 있는 봉사기프로쎄스들의 최대수량의 값을 조정하여야 한다. 체계는 또한 시동시에 8개의 프로쎄스를 시동한다.

우와는 별도로 관리자는 또한 httpd가 봉사할수 있는 의뢰기의 최대수량의 한계를 설정한다. 기정값은 150으로 설정된다.

```
MaxClients 150
```

HTTP는 TCP통신규약상에서 실행한다(TCP통신규약에서는 자료가 전송될수 있기전에 접속이 설정되여야 한다). 사용자가 련결상에서 찰칵할 때 새로운 접속이 설정되며 그에 의하여 파일전송시간이 증가된다. HTTP 1.1은 영구적인 접속을 유지하는 **생존**(keepalive)기능을 지원한다. 이 영구접속의 동작은 아래의 지령에 의하여 조종된다.

```
KeepAlive On
KeepAliveTimeout 15
```

두번째 설정은 다음요구가 도착할 때까지 현재의 접속을 15초동안 유지하도록 한다. 어떤 경우에는 모든 요구와 응답이 5분내로 완성되여야 한다.

```
Timeout 300
```

망에서의 혼잡은 주콤퓨터이름보다 오히려 IP주소를 사용하여 완화될수 있다.

```
HostnameLookups Off
```

이 설정은 수신접속이 DNS탐색을 호출하지 않으며 그에 따라 봉사기의 응답시간을 개선하도록 한다.

24.15 가상주콤퓨터관리

Web상의 모든 싸이트들이 다 독자적인 기계상에서 관리되는것은 아니다. Apache는 한 기계상에 **가상주콤퓨터**(virtual host)들을 지원하며 일반적으로 한개의 주콤퓨터가 1000개까지의 싸이트를 수용한다. 주콤퓨터는 여러개의 IP주소를 가질수 있으며 매 IP주소는 한개의 싸이트를 표시한다. 또한 하나의 IP주소가 여러개의 령역이름에 대응되게 할수 있다. 관리자는 <VirtualHost>지령을 가지고 모든 가상주콤퓨터를 위한 많은 봉사기파라메터를 설정할수 있다(이때 괄호를 주의).

Velvet는 Planets가 Web내용을 관리하기 위하여 자기들의 봉사기상에 제공하기로 한 20MB의 공간을 사용하기로 결정하였다. 싸이트 www.velvet.com은 자체의 개별적인 IP주소를 가지고 있다고 가정한다. 이 주소는 www.planets.com을 지정하는 DNS안에서 A레코드로서 규정되여야 한다. 한 묶음의 지령들이 <VirtualHost>와 </VirtualHost>꼬리표는 사이에서 개별적으로 정의될수 있다.

```
<VirtualHost www.velvet.com>
   ServerName www.velvet.com
   ServerAdmin webmaster@velvet.com
   DocumentRoot /home/velvet/www
   TransferLog /home/velvet/logs/access-log
   ScriptAlias /cgi-bin/ /home/velvet/www/cgi-bin/
</VirtualHost>
```

관리자는 첫 지령에 www.velvet.com대신에 IP주소를 사용할수 있다. 봉사기가 이 싸이트를 위한 요구를 수신하였을 때 ServerName을 www.planests.com 으로부터 www.velvet.com 으로 변화시킨다. 이 싸이트는 또한 자기의 DocumentRoot와 cgi-bin등록부를 가진다. Velvet는 Planets의 /etc/passwd안에 사용자등록자리 velvet를 가질것이다. 홈등록부기반의 주콤퓨터관리약속(hosting arrangement)을 가지는것에 의하여 Planets는 관리를 분산시킬수 있으며 Velvet가 자기의 싸이트를 관리하도록 할수 있다. 이것은 일반적으로 Velvet가 telnet나 ftp로 가상령역에 접속하기 위하여서는 허가권을 가져야 한다는것을 의미한다.

모든 가상주콤퓨터들에서 우의것과 같은 별개의 부분이 있어야 한다. Apache를 조정하는데 쓰이는 ServerRoot, ServerType와 같은 몇개의 례외들과 Apache조정에 사용된 파라메터들을 제외하고는 실천적으로 임의의 httpd.couf 지령이 여기서 사용될수 있다.

24.16 등록부접근조종

Apache는 access.conf를 통하여 문서들과 CGI프로그람들을 포함하는 등록부들에 대하여 접근제한을 설정할수 있으며 종종 설정에 리용되고 있다. 추가적으로 사용자들이 체계범위제약들을 재정의하게 한다. 접근제약은 <Directory>와 </Directory>꼬리표사이에 렬거된다. HTML파일들을 포함하는 등록부를 위한 일반적인 몇개의 제약항목들을 보면 아래와 같다.

```
<Directory /home/httpd/html>
  Options Indexes FollowSymLinks
  AllowOverride None              htaccess를 사용할수 없다
```

```
  order deny.allow
  deny from all
  allow from hillinfo.com
</Directory>
```

Options는 이 등록부와 그 보조등록부들을 위해 허가된 선택항목들을 지정한다. 봉사기는 파일 index.html을 발견할수 없으면 등록부안에 파일들의 목록을 만든다(Indexes). 또한 일반파일을 조작하는것과 꼭 같은 방법으로 기호련결을 사용한다(FollowSymLinks). AllowOverride는 사용자가 이 선택항목들을 재정의하도록 허용되는지를 결정한다. 여기서는 사용자가 할수 없지만(None) cgi-bin등록부는 모든 사용자가 이 선택항목들을 재정의하는것을 허가한다.

```
<Diretory /home/httpd/cgi-bin>
  AllowOverride All
  Options ExecCGI
</Directory>
```

AllowOverride All설정은 모든 사용자가 자기들의 국부파일 .htaccess안에 류사한 지령을 배치하는것으로 이 제약들을 무효화할수 있게 한다. 봉사기가 임의의 등록부로부터 파일을 검색할 때 먼저 이 파일을 찾아 본다. 만일 그것이 존재하고 기능재정의가 설정된다면 봉사기는 이 파일안에 명령문들을 배치할수 있다. 흔히 사용자들이 자기의 등록부들을 구성하도록 한다. 왜냐하면 그들이 .htaccess 파일을 변화시킬 때 봉사기가 재시동되지 말아야 하기때문이다.

cgi-bin등록부가 특수한 선택항목설정(ExecCGI)을 사용한다. ExecCGI설정을 가지고 있는 임의의 등록부는 CGI프로그람을 격납하는데 리용될수 있다. CGI프로그람은 Web상에서 큰 보안위험이며 특수한 등록부로부터 CGI프로그람을 실행시키기 위한 허가권이 암시적으로가 아니라 명백하게 설정되여야 한다. 그렇지만 같은 등록부를 참고하는 ScriptAlias문에 의하여 이미 그것이 실현되였으므로 그 명령문은 중복된다.

관리자가 등록부나무에 대한 접근권한을 설정하고 그다음 특수한 보조등록부를 위하여 그것들을 재정의할수 있다.

```
Alias /web/doc /home/html/docs

<Directory /web/docs>
    Options Indexes FollowSymLinks
    AllowOverride None
</Directory>

<Directory /web/docs/d1>
    AllowOverride All
</Directory>
```

봉사기는 /web/docs의 임의의 다른 보조등록부에서가 아니라 /web/docs/d1등록부안에서 .htaccess를 찾는다. 이것은 봉사기능률을 개선한다.

order, allow, deny문들은 이 등록부안에서 문서들을 접근하거나 프로그람들을 실행시키도록 허용

될수도 있는 주콤퓨터들을 정의하는데 사용된다. order는 접근규칙이 실현되는 순서를 결정한다. 여기서 deny규칙이 먼저 적용되고 그다음 allow규칙이 적용된다. 소수의 주콤퓨터들에만 접근을 허용할 때는 먼저 모든 사용자들의 권한을 금지시키고 그다음 일부 사용자들에게 접근을 허용하는것이 보통이다. 여기서는 hillinfo.com만이 그 등록부안의 파일들에 접근하도록 허락된다. 모두에게 접근을 허용하기 위해서는 allow from all을 사용한다.

주콤퓨터제한을 부과하는데서 Apache는 또한 등록부를 접근하기 위한 사용자인증을 요구할수 있다. 그때 사용자들은 어떤 봉사를 사용하기 위한 사용자통과암호를 제공해야 한다. Apache는 htpasswd프로그람을 사용하는 사용자인증체계를 가지고 있다. 발전된 기능이 일부 있는데 여기서는 설명하지 않는다.

참고

관리자가 특수한 등록부를 재정의하기 위한 임의의 봉사기선택항목들을 요구하지 않는다면 그 등록부상에 대하여 AllowOverride None을 사용할수 있다. Apache가 이 등록부의 어느 보조 등록부들에서도 .htaccess파일들을 탐색하지 않으므로 봉사기능률을 상당히 증가시킬수 있다.

요 약

인터네트는 FQDN들을 IP주소로, IP주소를 FQDN으로 분석하는 령역이름봉사(DNS)를 사용한다. 그 정보는 한 묶음의 이름봉사기(name server)들에 포함된다. 령역은 지역(zone)들로 나누어 지며 주봉사기(master server)는 그 지역에서 권한이 있다. 종속봉사기(slave server)는 지역전송(zonal tansfer)에 의하여 주봉사기로부터 그의 정보를 읽어 들인다.

질문처리는 TCP/IP응용프로그람안에 내장된 한 묶음의 서고루틴들인 분석기(resolver)에 의하여 수행된다. 그것은 파일 /etc/resolv.conf를 사용한다. 분석은 계층구조를 가지며 국부이름봉사기가 대답을 제공할수 없을 때에는 뿌리이름봉사기에 문의한다.

BIND는 DNS를 실현한 가장 일반적인 체계이며 named데몬을 사용한다. 암시파일(hints file)은 뿌리이름봉사기의 IP주소들과 FQDN들을 포함하며 모든 이름봉사기에 의하여 기억기안에 유지된다. 지역파일(zone file)은 IP주소-주콤퓨터이름대응표를 포함한다. 역탐색(reverse lookup)파일 및 국부주콤퓨터(localhost)파일들은 IP주소를 FQDN으로 변환하거나 127.0.0.1을 localhost로 변환하는 역분석을 수행한다. DNS는 또한 령역에 대한 우편을 수신하기로 되여 있는 봉사기들을 지정한다.

자료기지파일은 이름봉사기들의 IP주소를 지정하기 위하여 이름봉사기레코드(NS)를 사용하며 우편봉사기에로의 지시자들을 제공하기 위하여 우편교환기레코드(MX)를 사용한다. 주소레코드(A)와 지시자레코드(PTR)는 주소들을 FQDN으로 그리고 FQDN을 주소로 대응시킨다. 본명(canonical name)레코드(CNAME)는 별명을 제공하기 위하여 사용된다. 모든 파일들의 위치와 이름봉사기의 형은 /etc/named.conf안에 지정된다. ndc는 named를 개시하고 정지시킬수 있으며 한편 nslookup은 설정을 시험하는데 사용될수 있다.

Linux체계상에서 sendmail은 종합적인 전송대행체이며 procmail은 배포를 조작한다. 우편은 자주 전화선을 리용하여 POP(Post Office Protocol)규약으로 원격싸이트로부터 가져 온다.

중심집선기(hub) 즉 우편봉사기는 보통 모든 주콤퓨터를 대신하여 우편을 보내고 받는다. 그것은 종종 전자우편주소에 주콤퓨터이름을 나타내지 않도록 구성된다. 집선기는 개별적인 주콤퓨터들에 우편을 발송하거나 NFS로 설치된 파일체계를 제공할수도 있다. 그것은 또한 POP/IMAP봉사기로서도

동작한다.

sendmail은 우편을 수신하기 위하여서는 포구 25에서 데몬(-bd)으로서 실행되며 우편대기렬을 삭제하기 위하여서는 의뢰기(-q)로서 실행된다. 대기렬은 /var/spool/mqueue안에 유지된다. sendmail은 /etc/sendmail.cf로부터 지시를 가져 오며 /etc/aliases를 사용하여 우편발송을 수행한다. 별명(alias)은 한명이상의 사용자, 우편목록이나 파일에 우편을 다시 보내는데 사용된다.

sendmail.cf는 마크로(D)들을 사용하여 재치 있는 중계주콤퓨터(DS)를 정의하며 변장(DM)을 수행한다. 소문자마크로문자들이 내부적으로 정의된다. sendmail이 우편을 받는 령역을 정의하기 위하여 클라스가 사용된다(Cw). 선택항목(O)을 가지고 관리자는 sendmail이 실패를 통지하기전에 통보문의 배달을 시도해 볼 회수를 설정할수 있다(O Timeout.queuereturn).

Linux체계는 POP/IMAP봉사기로부터 우편을 얻기 위하여 fetchmail과 .fetchmailrc를 사용한다. 관리자는 통보문을 보관하겠는가(-k) 또는 그것들을 모두 회복하겠는가(-a)를 결정할수 있다. fetchmail은 하나의 우편통에 격납된 전체 령역에 대한 우편을 얻고 그다음 그것을 개별적인 우편통으로 배포하는데 사용된다.

Apache는 인터네트상에서 가장 널리 사용되는 Web봉사기이다. 그것은 포구 80에서 httpd데몬을 실행시키며 HTTP통신규약을 사용한다. httpd는 의뢰기요구를 조작하기 위하여 무권한방식에서 또 하나의 httpd프로쎄스를 생성한다. 구성은 httpd.conf안에서 설정되지만 많은 체계들은 srm.conf와 access.conf도 사용한다. Apache는 유연하며 실행시에 요구되는 모듈만을 읽어 들이도록 되여 있다(LoadModule과 AddModule).

구성파일은 봉사기의 뿌리등록부(ServerRoot)에 저장된다. HTML의 위치는 DocumentRoot에 의하여 설정된다. CGI프로그람들은 보안상의 리유로 별개의 등록부에 보관된다(ScriptAlias). 사용자들은 자기의 파일들을 개별적인 등록부들에 보관할수 있다(UserDir).

URL이 파일이름으로 끝나지 않을 때 다른 조건이 지정되지 않는한 봉사기는 파일 index.html을 탐색한다(DirectoryIndex). Apache와 현대의 열람프로그람들은 .Z와 .gz파일을 간단히 풀어 낼수 있다(AddEncoding). HTTP 1.1이 KeepAliveTimeout기간동안 영구적인 접속을 지원할수 있는것은 KeepAlive때문이다.

Apache는 여러개의 령역이 하나의 IP주소 또는 여러개의 IP주소들을 가지고 하나의 봉사기상에서 관리될수 있는 가상주콤퓨터관리(<VirtualHost>)를 지원한다. 모든 가상령역들은 문서위치들과 접근권한들을 위한 자체의 설정을 가질수 있다.

모든 등록부(<Directory>)는 자체의 접근제한도 가질수 있다. Options는 CGI프로그람들이 등록부로부터 실행하도록 허락하는가(ExecCGI) 또는 등록부목록을 허락하는가(Indexes)를 결정한다. 사용자가 이 선택항목들을 재정의하는것이 허용될수도 있고 허용되지 않을수도 있다(AllowOverride). 하나이상의 싸이트에 대하여 등록부에 대한 접근이 허용되거나(allow from) 금지(deny from)될수도 있다.

시험문제

1. 종속이름봉사기의 역할은 무엇이며 자기의 자료를 어떻게 얻는가?
2. ftp는 이름봉사기의 IP주소를 어떻게 얻는가?

3. DNS는 령역에 대한 우편을 수신하도록 되여 있는 우편봉사기들을 어떻게 지정하는가?
4. 하나의 주콤퓨터에 어떻게 여러개의 이름을 정의하는가?
5. 배포는 누가 하는가? 몇가지 중요한 배포프로그람들을 불러 보시오.
6. 나가는 우편은 어디에 보관되며 그 대기렬을 어떻게 보는가?
7. sendmail에 그것이 100개의 령역을 위한 우편을 수신해야 한다는것을 어떻게 통지하겠는가?
8. sendmail이나 POP가 체계상에서 실행하고 있는가를 어떻게 검사하겠는가?
9. /etc/aliases안에 발송을 정의하였는데 동작하지 않는다. 어떤 오유를 범하였을수 있는가?
10. httpd프로쎄스들이 체계에 몇개 있는가?
11. HTML문서를 위한 Web봉사기의 뿌리등록부를 어떻게 설정하는가?

련습문제

1. 령역과 지역사이의 차이점은 무엇인가? DNS가 그것들을 어떻게 조작하는가?
2. 기억전용이름봉사기란 무엇인가?
3. 뿌리이름봉사기들은 분석처리를 어떻게 방조하는가?
4. 언제 IP주소를 FQDN으로 변환해야 하는가?
5. 주봉사기항목들이 변경되였다는것을 종속봉사기가 어떻게 아는가?
6. user@hillinfo.com으로 주소화된 우편이 주콤퓨터 mail.hillinfo.com에 의하여 조종된다. 그러나 어느 주콤퓨터가 user@www.hillinfo.com으로 주소화된 우편을 받는가?
7. telnet edison.hillinfo.com외에 telnet edison을 가지고 싸이트를 접근할수 있다면 그 리유는 무엇인가? /etc/hosts안에 항목이 없다고 가정하라.
8. 30분마다 우편을 수신하고 대기렬을 삭제하기 위하여서는 sendmail을 어떻게 호출하겠는가?
9. SMTP가 또 다른 SMTP에 직접 우편을 배달하는데 왜 우편이 사람들에게 전달되는데 이따금 몇일씩이나 걸리는가?
10. 우편을 발송할 때 자기의 주콤퓨터이름을 나타내지 않으며 ISP가 자기의 우편을 조작하게 하려면 어떻게 해야 하는가?
11. 우편이 배포되지 않았다는것을 한시간내로 통지하려고 한다. 어느 설정을 변화시켜야 하는가?
12. Netscape Messenger를 리용하는 의뢰기가 집선기에 우편을 보내기 위하여 sendmail을 어떻게 호출해야 하는가?
13. 수신우편봉사기에서 /etc/passwd의 역할은 무엇인가?
14. 사용자 melinda가 자기의 홈등록부에 위치한 Web페지에 접근하도록 하자면 Apache를 어떻게 구성해야 하는가? 그 등록부는 URL로 어떻게 접근되는가?
15. URL이 파일이름을 포함하지 않을 때 어떻게 Apache가 파일 default.html을 봉사하게 하겠는가?
16. 영구적인 접속을 관리하도록 하려면 봉사기를 어떻게 구성해야 하는가?
17. ScriptAlias문의 의미는 무엇인가?
18. 령역 velvet.com으로부터 cgi-bin등록부에 대한 접근을 불가능하게 하려면 어떻게 해야 하는가?
19. 등록부상에서 CGI실행을 가능하게 하려면 어떻게 해야 하는가?

부록 1. C쉘프로그람작성구조

C쉘은 vi의 설계자인 윌리암 죠이(William Joy)가 캘리포니아종합대학 버클리에서 개발하였다. Korn쉘과 bash는 둘 다 이 쉘의 많은 기능들을 도입하였다. Linux는 대단히 우월한 C쉘인 tcsh를 제공한다. 우리는 제17장에서 이 쉘의 환경관련특성들을 취급하였으며 이 부록에서는 그의 프로그람작성구조들을 취급한다.

계산

C쉘은 옹근수를 처리할수 있다. 변수값주기는 set 또는 @로 진행한다.

```
% set x=5
% @ y = 10
% @ sum=$x + $y
% @ product = $x * $y
% @ quotient = $y/$x
@: Badly formed number
% @ quotient = $y / $x
```

수의 증가는 아래의 방법으로 진행된다.

```
@ x = $x + 1
@ x++
@ x ++
```

같기표(=)가 그 주위에 어떤것도 요구하지 않는다고 해도 @의 뒤에는 공백이 놓여야 한다. 또한 연산자(+, -, *, /, %기호)의 량쪽에 공백이 있어야 한다. 변수값은 보통 echo지령으로 현시한다.

배렬

C쉘은 여러개의 변수들을 가지는데 일부는 자체에 유일한것이며 일부는 다른 쉘들의것과 류사한것들이다. 제17장에서 이 변수들을 어떻게 설정하고 어떻게 값을 구하는가를 배웠다. 그러나 어떤 변수(path)는 좀 류다르게 설정된다는것을 보았을것이다.

```
set path = (/bin /usr/bin /usr/local/bin /usr/dt/bin .)
```

우의 배렬은 실제적으로 이 쉘에서 지원하는 배렬(array)이다. path는 보통변수로서 참조되지만 5개의 원소를 가진 배렬로도 간주될수 있다. 첫 원소는 $path[1]로, 둘째 원소는 $path[2]로 접근된다. 배렬에서 원소의 개수는 파라메터 $#로서 가리키며 그뒤에 변수(또는 배렬)이름이 놓인다. 실례를 몇가지 들어 보자.

```
% echo $path
/bin /usr/bin /usr/local/bin /usr/dt/bin .
% echo $path[4]
/usr/dt/bin
% echo $#path
5
```

set명령문을 가지고 배렬안에 값을 넣을수 있으며 shift도 배렬과 함께 작업할수 있다.

```
% set numb = ( 9876 2345 6213 )
% echo $numb[1]
```

```
9876
% echo $#numb
3
% shift numb
% echo $numb[1]
2345
```

스크립트를 실행시키기

기정적으로 C쉘언어로 씌여 진 스크립트들은 Bourne쉘에 의하여 실행된다(Linux는 tcsh스크립트를 실행시키기 위하여 bash를 사용한다). 이것을 하기 위한 두가지 방법이 있다. 즉 csh지령을 가지고 스크립트를 실행시키든지 아니면 모든 C쉘스크립트의 시작에 해석기(interpreter)기능을 두는것이다.

```
csh script_name
#! /bin/csh
```

대부분의 C쉘들은 보다 편리한 두번째 형식을 지원하지만 자기의 판본이 그것을 지원하지 않는다면 선택항목은 csh지령을 사용한다.

if문

여기서 if문은 다른 형식을 가진다. 먼저 열쇠단어 then이 if와 같은 행에 있어야 한다. 둘째로 명령문종단기호는 fi가 아니라 endif이다. 끝으로 검사되여야 하는 조건은 보통 한쌍의 괄호로 닫겨 져야 한다.

```
% cat filesize.csh
#!/bin/csh
# Program: filesize.csh - Converts file blocks to size in bytes
    if ( $#argv != 2 ) then              # -ne대신에 !=로 검사되는 조건
    echo Two parameters required
else
    @ size = $1 * 512
    echo Size of $2 is $size bytes
endif
% filesize.csh 124 tulec04
Size of tulec04 is 63488 bytes
```

수값비교는 Bourne이 사용하는것(-gt, -lt 등)보다 오히려 C형의 연산자 >, ==, != 등으로 이루어 진다. #$argv는 스크립트에로 넘겨 진 인수의 개수로 설정된다. 인수들은 $argv[1], $argv[2] 등으로 개별적으로 접근된다. 다른 쉘들과의 호환성을 위하여 C쉘은 $1, $2 등으로도 그것들에 접근할수 있게 한다.

lse문이 없을 때에는 한개의 행으로 조건식을 꾸밀수 있다.

```
if ( $#argv == 2 ) @ size = $1 * 512 ; echo Size of $2 is $size bytes
```

UNIX지령을 조종지령으로 실행시킬 때에는 지령 그자체가 한조의 대괄호에 의하여 닫겨 져야 한다.

```
if { grep "director" emp.lst } then
```

끝으로 이 쉘에는 test문이나 그의 동의어 []가 없다는것을 기억하시오. 그렇지만 여기서 파일속성의 일부는 test에 적용될수 있다. 파일 foo가 읽기가능한가를 시험하기 위하여 if (-r foo)를 사용할수 있으며 bar가 등록부가 아닌가를 검사하기 위하여서는 if (! -d bar)를 사용할수 있다. 파일시험의 완전한 목록은 부록 4에 보여 주었다.

switch문

switch문은 하나이상의 갈래를 위하여 식을 비교하며(case와 같이) 열쇠단어 endsw, case, breaksw를 사용한다. 아래에 간단한 실례로서 3-선택항목차림표를 보여 주었다.

```
% cat menu.csh
#!/bin/csh
set choice = $argv[1]              # 인수로 제공되는 선택항목
switch ($choice)
case 1:
    ls -l
    breaksw              # 정합을 중지한다
case 2:
    ps -f
    breaksw
case 3:               # 마지막항목에 대해서는 default:을 쓸수 있다
    exit
    breaksw
endsw
```

여기서 case문은 좀 다르게 리용되며 두점으로 끝난다. breaksw는 대응되는 값이 발견되고 비교작용이 수행된후에 그 구조체의 조종을 중지시킨다. 이 단어가 제공되지 않으면 모든 선택항목들에 해당한 동작이 수행된다. 이것은 거의 모든 경우에 리치에 맞지 않으므로 그것이 거기에 있는가를 확인하시오. 흔히 마지막의 case선택항목은 일부 동작(보통 프로그람을 중지시키는)을 수행하는데 리용된다. 그러한 경우에 변수 choice의 특수한 값이 문제로 되지 않으며 default를 사용할수 있다.

while 및 foreach순환

두개의 순환 즉 while과 foreach(for대신)가 있다. 두 순환은 다른 쉘과 비교할 때 3가지 주요한 차이점을 가진다.

- 순환조건(또는 그 목록)이 괄호로 닫겨 진다.
- do열쇠단어가 사용되지 않는다.
- 순환은 done대신에 end로 끝난다.

먼저 while순환을 고찰하자. 프롬프트에 간단한 렬을 입력한다.

```
% set x = 5
% while ( $x > 3 )
?  ps -f
?  sleep 5
? end
```

foreach도 Bourne의것과 차이점을 가지지만 perl이 그것을 모의하였다. 열쇠단어 foreach는 for를 대신하며 열쇠단어 in이 필요 없다. 18.16에서의 실례가 다음과 같이 고쳐 질수 있다.

```
% foreach file (chap20 chap21 chap22 chap23)
?       cp $file ${file}.bak
?       echo $file copied to $file.bak
? end
```

목록을 사용하는 다른 방법들이 있다.

```
foreach item ( `cat clist` )
foreach fname ( *.c )
foreach fname ( $* )
```

repeat문

한개의 지령이 유한수만큼 반복되여야 한다면 repeat문을 사용할수 있다.

```
% repeat 3 date
Mon Jan 17 22:40:52 EST 2000
Mon Jan 17 22:40:52 EST 2000
Mon Jan 17 22:40:52 EST 2000
```

goto문

goto문도 사용할수 있다. 오늘날에는 프로그람개발자들이 거의나 사용하지 않지만 주의하여 사용하면 기교 있게 빠져 나가는 좋은 방법이 된다.

```
% cat gotoexamp.csh
#!/bin/csh
if ( $#argv == 0 ) then              #입력된 인수가 없다면
    goto endblock
else
    grep $1 emp.lst
    exit
endif
endblock:
echo "You have not keyed in an argument"
```

exit문은 grep가 실행을 완료한후에 조종이 실패로 끝나지 않고 echo지령의 실행에서 끝났는가를 확인한다. 만일 거기에 exit문을 배치하지 않으면 인수가 제공되였는가 아닌가에 상관없이 오유통보문이 나타날것이다.

onintr문

onintr문(다른 쉘에서는 trap)은 새치기신호가 스크립트에 가해 졌을 때 실행될 지령들을 지정한다. 보통 쉘스크립트의 시작에 위치한다.

```
% cat onintr.csh
#!/bin/csh
onintr cleanup
cut -c1-10 index > $$
cut -c21- index > $$.1
paste $$ $$.1 > pastelist
rm $$ $$.1
exit

cleanup:
rm $$ $$.1
echo "Program interrupted"
```

goto문과 같이 onintr문의 뒤에는 표식(label)이 뒤따른다. 새치기건이 눌리울 때 실행은 이 꼬리표에로 이행된다. 이 신호를 무시하고 처리를 계속하려 할수도 있는데 그 경우에는 onintr를 ?와 함께 써서 프로그람이 그러한 신호의 영향을 받지 않도록 하여야 한다.

```
onintr -
```

더 강력한 프로그람작성구조를 가진 Korn쉘과 bash에 의하여 능력과 기능에서 C쉘은 완전히 교체되였다. 만일 개선된 C쉘을 찾고 있다면 Linux에서 제공되는 tcsh를 사용하시오.

부록 2. vi/vim과 emacs지령참고서

매 편집기는 자기의 고유한 특징을 가지고 있는 반면에 그 기능에서 상당한 중복을 가지고 있다. 이 부록은 vi/vim과 emacs편집기들의 편집특성을 비교하여 설명한다. dired기능이나 우편조작기능과 같은 emacs의 비편집기능은 제외한다. 여기서 많은 지령들은 반복인자(repeat factor:vi)나 수자인수(digit argument:emacs)와 함께 사용될수 있다. 구역에서 동작하는 지령들은 emacs와 vim에만 적용될뿐 vi에는 적용되지 않는다. 구체적인것은 이 지령들의 대부분이 설명된 제4장과 제5장을 보시오. 편리상 지령들과 그것들의 습관적인 파라메터들을 함께 묶었다. 문장과 단락에서 동작하는 지령들은 고찰하지 않았다.

이 부록에서 완전히 단어화한 emacs지령들이 나타나면 그것은 [Alt-x]를 앞에 놓아야 하는데 그에 대해서는 명백하게 지정하지 않았다. 그러나 set-variable과 함께 설정할 필요가 있는 변수들은 보여 주었다. set-variable과 함께 사용된 모든 변수들은 또한 setq지령을 사용하는 .emacs안에 놓일수 있다.

여기서 [Meta]건을 나타내기 위하여 [Alt]건이 emacs와 함께 사용되였다. 체계상에서 실제적으로 동작하는 건([Esc]와 같은)을 사용하시오.

항행 (Navigation)

vi지령	기 능	emacs지령
h (or [Backspace])	유표를 왼쪽으로 이동시킨다	[Ctrl-b]
l (or [Spacebar])	유표를 오른쪽으로 이동시킨다	[Ctrl-f]
k (or [Ctrl-p])	유표를 우로 이동시킨다	[Ctrl-p]
j (or [Ctrl-n])	유표를 아래로 이동시킨다	[Ctrl-n]
[Ctrl-f]	한페지 앞으로 옮긴다	[Ctrl-v]
[Ctrl-b]	한페지 뒤로 옮긴다	[Alt-v]
[Ctrl-d]	반페지 앞으로 옮긴다	—
[Ctrl-u]	반페지 뒤로 옮긴다	—
1G	파일의 시작으로 이동한다	[Alt-<]
40G	40행으로 이동한다	goto-line 40
G	파일의 끝으로 이동한다	[Alt->]
-	행번호현시방식을 절환한다	line-number-mode
[Ctrl-g]	현재행번호와 파일의 비률을 현시한다	what-line
:set number (nu)	모든 행들에 번호를 붙여 보여 준다	—

행에서의 항행

vi에서의 B, E, W지령들은 반점이 무시된다는것을 제외하고는 소문자지령과 같은 기능을 수행한다. emacs에는 대응한 반점무시기능이 없다.

vi지령	기 능	emacs지령
b	단어의 시작으로 후진한다	[Alt-b]
e	단어의 끝으로 전진한다	[Alt-f]
w	단어의 시작으로 전진한다	—
0 or |	행의 시작으로 이동한다	[Ctrl-a]
30|	30렬로 이동한다	[Ctrl-a][Alt-30][Ctrl-f]
^	행에서 첫 단어의 첫 문자로 이동한다	[Alt-m]
$	행의 끝으로 이동한다	[Ctrl-e]

본문삽입

vi와 달리 emacs는 항상 입력방식이다. 여기서 보여 준 emacs지령들은 단순히 행에서 삽입을 시작할수 있는 특정한 점으로 옮겨 준다. emacs에서 본문의 삽입은 text로서 보여 주었다. 여기서 조종문자의 삽입은 [Ctrl-b]로 보여 주었다.

vi지령	기 능	emacs지령
i	유표의 왼쪽에 본문을 삽입한다	text
20i-[Esc]	20개의 ?기호를 삽입한다	[Alt-20]-
I	행의 시작에 본문을 삽입한다	[Ctrl-a] text
a	유표의 오른쪽에 본문을 추가한다	[Ctrl-f] text
A	행의 끝에 본문을 추가한다	[Ctrl-e] text
o	행을 아래로 펼친다	[Ctrl-e][Enter]
O	행을 우로 펼친다	[Ctrl-a][Enter]
[Ctrl-v][Ctrl-b]	[Ctrl-b]를 삽입한다	[Ctrl-q][Ctrl-b]
[Ctrl-v][Esc]	[Esc]를 삽입한다	[Ctrl-q][Esc]
:set showmode	vi가 입력방식일 때 통보문을 현시한다	—
:set sm	)와 }에 대한 짝을 즉시 보여 준다	blink-matching parent(with set- variable)
:set ts=n	타브걸음을 n으로 설정한다(초기값:8)	edit-tab-stops
:set ai	다음행이 이전 행의 첫 렬위치에서 시작한다	—

본문의 삭제와 이동

이 부분의 모든 편집동작들은 취소될수 있다. 그렇지만 emacs에서 한 문자를 삭제하는 지령인 [Ctrl-d]는 회복조작을 제공하지 않는다. 즉 [Ctrl-d]로 지워 진 문자는 제거고리로부터 회복될수 없다. 그러나 그 지령이 수자인수의 앞에 놓일 때에는 그 삭제가 회복될수 있다.

vi지령	기 능	emacs지령
x	유표아래의 문자를 삭제한다	[Ctrl-d] 혹은 [Delete]
6x	유표아래의 문자와 오른쪽 5개 문자를 삭제한다	[Alt-6][Ctrl-d]
X	앞문자를 삭제한다	—
dd	현재행을 삭제한다	[Ctrl-a][Ctrl-k][Ctrl-k]
4dd	4개 행을 삭제한다	[Alt-4][Ctrl-k]
64dd	64개 행을 삭제한다	[Ctrl-u][Ctrl-u][Ctrl-u][Ctrl-k]
dw	단어를 삭제한다	[Alt-d]
—	앞단어를 삭제한다	[Alt][Delete] 혹은 [Alt][Backspace]
d0(d와 령)	행의 시작까지 지운다	[Alt-0][Ctrl-k]
d$	행의 끝까지 지운다	[Ctrl-k]
—	빈 행들을 삭제한다	[Ctrl-x][Ctrl-o]
d	구역을 삭제한다(vim에서만)	[Ctrl-w]
p	오른쪽에 삭제된 본문을 넣는다	[Ctrl-y]
P	우에 또는 왼쪽에 삭제된 본문을 넣는다	—
"add	현재행을 완충기 a에로 삭제하여 넣는다	[Ctrl-u][Ctrl-x] xa(구역에서)
"ap	완충기 a로부터 내용을 회복한다	[Ctrl-u][Ctrl-x] ga
—	제거고리에 삭제/복사될 본문을 위한 n개의 단락을 보관한다	kill-ring-max n (with set-variable)
ddp	현재행을 다음행과 바꾼다	[Ctrl-n][Ctrl-x][Ctrl-t]

vi지령	기 능	emacs지령
kddp	현재행을 이전 행과 바꾼다	[Ctrl-x][Ctrl-t]
J	현재행을 다음행과 련결한다	[Ctrl-e][Ctrl-d]
kJ	현재행을 이전 행과 련결한다	[Alt-^]
xp	두 문자를 바꾼다	[Ctrl-t]
—	두 단어를 바꾼다	[Alt-t]
—	행을 중심으로 옮긴다	center-line

본문의 복사

vi지령	기 능	emacs지령
yy	현재행을 복사한다	구역을 요구한다
6yy	6개 행을 복사한다	우와 같다
yw	단어를 복사한다	우와 같다
y	구역을 복사한다(vim에서만)	[Alt-w]
p	복사된 본문을 오른쪽에 놓는다	[Ctrl-y]
P	복사된 본문을 왼쪽 또는 우에 놓는다	—
"ayy	현재행을 완충기 a에 복사한다	[Ctrl-x] xa
"ap	완충기 a로부터 내용을 회복한다	[Ctrl-x] ga

본문의 변경, 읽어들이기, 려과

vi지령	기 능	emacs지령
rch	유표아래의 한 문자를 ch로 바꾼다	[Ctrl-d] ch
R	유표로부터 오른쪽으로 본문을 교체한다	overwrite-mode, text
s	유표아래의 한 문자를 임의의 수의 문자들로 교체한다	[Ctrl-d] text
S	전체 행을 교체한다	[Ctrl-a][Ctrl-k] text
cw	단어를 고친다	[Alt-d] text
c	구역의 본문을 고친다(vim에서만)	
~	주사한 본문이나 구역의 대소문자를 반전시킨다	
—	단어를 대문자로 변환한다	[Alt-u]
—	단어를 소문자로 변환한다	[Alt-l]
—	단어의 첫 문자를 대문자로 만든다	[Alt-c]
!navigation_cmd cmd	navigation_cmd로 끝나는 구역이나 부분에서 지령 cmd를 실행시킨다	[Ctrl-u][Alt-l] cmd
—	구역상에서 cmd를 실행시키며 분리된 창문에 출력한다	[Alt-l] cmd
!navigation_cmd sort	navigation_cmd로 끝나는 구역이나 부분을 정돈한다	sort-lines
!tr '[a-z]' '[A-Z]'	구역을 대문자로 변환한다(vim에서만)	[Ctrl-x][Ctrl-u]
!tr '[A-Z]' '[a-z]'	구역을 소문자로 변환한다(vim에서만)	[Ctrl-x][Ctrl-l]
:r foo	현재행의 아래에 파일 foo를 읽어 들인다	[Ctrl-x]i foo
:r !head -3 foo	foo의 첫 3개 행을 현재행의 아래에 읽어 들인다	[Ctrl-u][Alt-!]head -3 foo

편집기를 시동시키기

vi지령	기 능	emacs지령
vi +100 foo	200행에서 파일을 연다	emacs +100 foo
vi +/pat foo	패턴 pat가 처음 나타나는 곳에서 파일을 연다	—

vi지령	기 능	emacs지령
vi + foo	파일을 끝에서 연다	—
—	henry의 .emacs를 읽어 들인다	emacs -u henry foo
—	.emacs를 읽어 들이지 않는다	emacs -q foo
vi -R foo	파일을 읽기전용방식으로 연다	

보존 및 탈퇴

vi지령	기 능	emacs지령
:w	파일을 보관하고 편집방식을 유지한다	[Ctrl-x][Ctrl-s]
:w bar	Microsoft Windows의 Save As …와 같다	[Ctrl-x][Ctrl-w]
:w! bar	우와 같으나 존재하는 파일 bar에 덧쓰기한다	—
:n1,n2w foo	행 n1부터 n2까지를 파일 foo에 써넣는다	write-region (on region)
:n1,n2w >> foo	행 n1부터 n2까지를 파일 foo에 추가한다	append-to-file (on region)
:.w foo	현재행을 파일 foo에 써넣는다	—
:$w foo	마지막행을 파일 foo에 써넣는다	—
:x	파일을 보관하고 편집방식에서 탈퇴한다	[Ctrl-u][Ctrl-x][Ctrl-c]
:wq	우와 같다	
:q	파일을 보관함이 없이 편집방식에서 탈퇴한다	[Ctrl-x][Ctrl-c]
:q!	변경을 포기하고 편집방식에서 탈퇴한다	우와 같으나 n을 입력하고 프롬프트상에서 yes를 누른다
—	자동보관방식을 가능/불가능하게 한다	auto-save-default (t or nil)(with set-variable)
—	자동보관간격을 n개 건입력으로 설정한다	auto-save-interval n(with set-variable)
—	자동보관간격을 n초로 설정한다	auto-save-timeout n(with set-variable)
—	자동보관된 파일을 회복한다	recover-file

여러개의 파일에 대한 편집

vi에서는 현재파일이 보관되지 않는한(그리고 autowrite가 설정되지 않는한) :e, :n, :rew를 리용할수 없다. 지령의 뒤에 놓인 !는 안전기능을 재정의한다. emacs에서는 한 파일로부터 다른 파일로 보관함이 없이 자유롭게 옮겨 갈수 있게 한다.

vi지령	기 능	emacs지령
:e foo	현재파일의 편집을 중지하고 파일 foo를 편집한다	[Ctrl-x][Ctrl-f]
:e! foo	우와 같으나 먼저 현재파일의 변경을 포기한다	보존이 필요 없다
—	현재의 완충기를 다른 파일로 교체한다	[Ctrl-x][Ctrl-v]
:e!	현재파일의 마지막보관내용을 읽어 들인다	revert-buffer
[Ctrl-^]	가장 최근에 편집된 파일로 돌아 간다	[Ctrl-x] b [Enter]
:n	다음파일을 편집한다	[Ctrl-x] b ,목록으로부터 선택한다
:set autowrite (aw)	파일들을 절환할 때마다 현재파일을 자동적으로 써넣는다(vi에서는 :n과 함께)	필요 없다
:rew	첫 파일의 편집을 시작하기 위하여 파일목록을 다시 감는다	—
—	현재완충기를 제거한다	[Ctrl-x] k

다중창문

지 령	기 능	emacs지령
:sp	현재창문을 두개로 가른다	[Ctrl-x] 2
:new	새로운 빈 창문을 연다	[Ctrl-x] b
[Ctrl-w][Ctrl-w]	창문들사이를 절환한다	[Ctrl-x] o
:on	현재창문을 유일창문으로 한다	[Ctrl-x] 1
:q	현재창문에서 탈퇴한다	[Ctrl-x] 0(령)
:qa	모든 창문에서 탈퇴한다	
:xa	모든 창문을 보관하고 탈퇴한다	[Ctrl-x][Ctrl-c] 다음 !
—	다른 창문에서 본문을 앞으로 이동시킨다	[Ctrl][Alt] v
[Ctrl-w] +	창문크기를 증가시킨다	[Ctrl-x] ^
[Ctrl-w] -	창문크기를 줄인다(emacs에서는 n개 행만큼)	[Ctrl-u] -n [Ctrl-x] ^
	파일을 또 다른 창문에 연다	[Ctrl-x] 4 [Ctrl-f]

검색과 반복

emacs와 달리 vi는 문자렬과 정규식에 같은 검색 및 반복기능을 사용한다. emacs는 검색을 증가적으로 할수도 있다. vi는 행안에서의 문자의 검색에 별도의 건조작을 사용한다.

vi지령	기 능	emacs지령
/pat	문자렬 pat에 관하여 비증가적인 전진검색을 진행한다	[Ctrl-s][Enter] pat
/pat	우와 같으나 pat는 정규식이다	[Ctrl][Alt] s [Enter] pat
?pat	문자렬 pat에 관하여 비증가적인 후진검색을 진행한다	[Ctrl-r][Enter] pat
?pat	우와 같으나 pat는 정규식이다	[Ctrl][Alt] r [Enter] pat
—	문자렬 pat에 관하여 증가적인 전진검색을 진행한다	[Ctrl-s] pat
—	우와 같으나 pat는 정규식이다	[Ctrl][Alt] s pat
—	문자렬 pat에 관하여 증가적인 후진검색을 진행한다	[Ctrl-r] pat
—	우와 같으나 pat는 정규식이다	[Ctrl][Alt] r pat
n	같은 방향/전진방향에서 문자렬검색을 반복한다	[Ctrl-s]
N	반대방향/후진방향에서 문자렬검색을 반복한다	[Ctrl-r]
n	같은 방향/전진방향에서 정규식검색을 반복한다	[Ctrl][Alt] s [Enter][Enter]
N	반대방향/후진방향에서 정규식검색을 반복한다	[Ctrl][Alt] r [Enter][Enter]
—	유표뒤의 단어를 검색한다	[Ctrl-s][Ctrl-w]
—	검색을 취소한다	[Esc]
:set wrapscan (ws)	파일의 다른 끝으로 이동하여 패턴검색을 계속한다	—
:set ignorecase (ic)	검색시에 대소문자를 구별하지 않는다	case-fold-search t(with set-variable)
:set magic	정규식문자들의 의미를 얻는다	
fc	문자 c에 관하여 전진검색을 진행한다	[Ctrl-s] c
Fc	문자 c에 관하여 후진검색을 진행한다	[Ctrl-r] c
;	문자에 관한 마지막전진검색을 반복한다	[Ctrl-s]
,	문자에 관한 마지막후진검색을 반복한다	[Ctrl-r]

치환

vi지령	기 능	emacs지령
:1,$s/s1/s2/g	전반적으로 문자렬 s1을 s2로 교체한다	replace-string
:1,$s/s1/s2/g	우와 같으나 s1은 정규식이다	replace-regexp

vi지령	기 능	emacs지령
:1,$s/s1/s2/gc	대화적인 교체를 진행한다	[Alt-%]
:1,$s/s1/s2/gc	우와 같으나 s1은 정규식이다	query-replace-regexp
	치환시에 문자들의 원래의 대소문자를 보관한다	case-replace t(with set-variable)
:s	현재행에서 마지막치환을 반복한다(vim에서만)	

표식과 서표(bookmark)

vi지령	기 능	emacs지령
ma	표식 a를 설정한다	[Ctrl-x]/a
'a	표식 a에로 이동한다	[Ctrl-x] ja
' '	현재위치와 이전 위치사이를 절환한다	[Ctrl-x][Ctrl-x]
—	서표를 설정한다	[Ctrl-x] rm
—	서표에로 이동한다	[Ctrl-x] rb

재실행(redo) 및 취소(undo)

vi지령	기 능	emacs지령
.	마지막지령을 반복한다([Alt-x]를 가진 emacs지령)	[Ctrl-x][Esc][Esc]
u	마지막편집지령을 취소한다	[Ctrl-x]u 혹은 [Ctrl--]
[Ctrl-r]	마지막취소를 다시 수행한다(vim에서만)	모든 취소를 수행한 다음에 우와 같은 동작을 진행한다
U	현재행에 만들어 진 모든 변경을 취소한다	—
"4p	완충기로부터 4번째의 최근 삭제를 회복한다(vim에서만)	[Ctrl-u] 4 [Ctrl-y]
u. after initial "1p	이전 회복을 취소, 다음완충기로부터 본문을 회복	[Alt-y] after initial [Ctrl-y]

본문생략

vi지령	기 능	emacs지령
—	생략기능을 가능하게 한다(emacs에서는 쌍방향절환)	abbrev-mode
:ab stg name	name을 stg로 생략한다	stg [Ctrl-x] aig name (ail for local abbreviation)
—	완충기안의 유효한 문자렬로부터 략어를 확장한다	[Alt-/]
:ab	모든 략어를 렬거한다	list-abbrevs
:unab stg	략어 stg를 제거한다	Delete abbreviation with edit-abbrevs
—	모든 략어를 제거한다	kill-all-abbrevs
—	현재의 모든 략어를 보관한다	write-abbrev-file
—	략어를 포함한 파일을 읽어 들인다	read-abbrev-file
—	앞으로의 략어들을 모두 보관한다	save-abbrevs t(with set-variable)

마크로와 건대응

vi지령	기 능	emacs지령
지령렬을 입력한후 "ayy	마크로를 정의한다(vi에서는 a라고 이름 짓는다)	[Ctrl-x] (commands [Ctrl-x])
@a	마지막으로 정의된 마크로를 실행한다	[Ctrl-x] e
—	마지막으로 정의된 마크로를 macroname으로 이름 짓는다	name-last-kbd-macro
@m	마크로 m(vi) 또는 macroname(emacs)를 실행시킨다	macroname (with [Alt-x])

vi지령	기 능	emacs지령
:map key commands	key를 commands(vi) 또는 macroname (emacs)과 대응시킨다	global-set-key,건과 마크로이름을 입력하시오
:map! key commands	입력방식에서 key를 commands로 대응시킨다	우와 같다
.exrc에 :map지령을 놓는다	마크로를 파일에 보관한다	insert-kbd-macro
—	마크로를 포함한 파일을 읽어 들인다	load-file
:map	지령방식대응을 모두 현시한다	—
:map!	입력방식대응을 모두 현시한다	—
:unmap key	지령방식대응 key를 제거한다	—
:unmap! key	입력방식대응 key를 제거한다	—

UNIX와의 결합부

편집기는 일감조종을 가능하게 하는 쉘들에서만 [Ctrl-z]로 중지시킬수 있다.

vi지령	기 능	emacs지령
:!cmd	UNIX지령 cmd를 실행한다	[Alt-!] cmd
:!%	현재의 파일을 쉘이나 perl스크립트로 실행시킨다	No shortcut
:sh	UNIX쉘에서 탈퇴한다	shell
[Ctrl-z]	편집기를 중지시킨다	[Ctrl-z] or [Ctrl-x][Ctrl-z]
:!cc %	현재 편집되는 C프로그람을 콤파일한다	compile
:!javac %	현재 편집되는 Java프로그람을 콤파일한다	No shortcut

도움말(emacs에서만 유효)

vi지령	기 능	emacs지령
—	건입력에 의하여 수행되는 기능(상세히)	[Ctrl-h] k
—	건입력에 의하여 수행되는 기능(한행정도)	[Ctrl-h] c
—	지령에 의하여 수행되는 기능	[Ctrl-h] f
—	지령에 유효한 건묶음	[Ctrl-h] w
—	변수와 현재설정에 대한 기능	[Ctrl-h] v
—	개념을 사용하는 지령들	[Ctrl-h] a
—	훈련소개프로그람(tutorial)을 실행시킨다	[Ctrl-h] t
—	정보열람프로그람(info reader)을 실행한다	[Ctrl-h] i

기타

vi지령	기 능	emacs지령
—	입력렬을 취소한다	[Ctrl-g]
:set all	설정된 선택항목들(vi)이나 변수들(emacs)을 모두 보여 준다	list-options
[Ctrl-l]	화면을 다시 그린다	[Ctrl-l]
v	구역의 시작을 정의한다(vim에서만)	[Ctrl-@] 혹은 [Ctrl][Spacebar]
—	[Alt-x]와 함께 사용된 문자렬 stg로부터 지령을 완성한다	stg [Tab]
:set ro	읽기전용방식으로 바꾼다(vi), 방식을 절환한다(emacs)	[Ctrl-x][Ctrl-q]

부록 3. 정규식묶음

편집기 vi/vim과 emacs, 려파기 grep, egrep, sed, awk, perl은 정규식(regular expression)을 사용한다. 결국 그것들은 이 자료의 서로 다른 부분집합을 사용하며 거의 대부분의 사람들은 지령이 인식하는 메타문자들과 인식하지 못하는 메타문자들을 확실하게 알지 못하고 있다. 이 부록은 매 지령이 가지고 모든 메타문자들을 시험해 본후에 준비되였다. 리해를 쉽게 하도록 하기 위하여 때때로 실례들을 제공하였다.

기초정규식들

								Linux				
기호	vi	emacs	grep	sed	egrep	awk	perl	vim	grep	sed	gawk	정 합
*	•	•	•	•	•	•	•	•	•	•	•	령개이상의 앞문자
g*	•	•	•	•	•	•	•	•	•	•	•	아무것도 없거나 g, gg, ggg 등
gg*	•	•	•	•	•	•	•	•	•	•	•	g, gg, ggg, 등
.	•	•	•	•	•	•	•	•	•	•	•	한 문자
.*	•	•	•	•	•	•	•	•	•	•	•	령 혹은 어떤 개수의 앞문자
[abc]	•	•	•	•	•	•	•	•	•	•	•	a 또는 b 또는 c
[1-3]	•	•	•	•	•	•	•	•	•	•	•	1과 3사이의 수자
[^Z]	•	•	•	•	•	•	•	•	•	•	•	Z를 제외한 임의의 문자
[^a-zA-Z]	•	•	•	•	•	•	•	•	•	•	•	영문자가 아닌 문자
^DM	•	•	•	•	•	•	•	•	•	•	•	행의 시작에 DM
bash$	•	•	•	•	•	•	•	•	•	•	•	행의 끝에 bash

확장된 정규식들

<table>
<tr><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th colspan="4">Linux</th><th></th></tr>
<tr><th>기호</th><th>vi</th><th>emacs</th><th>grep</th><th>sed</th><th>egrep</th><th>awk</th><th>perl</th><th>vim</th><th>grep</th><th>sed</th><th>gawk</th><th>정 합</th></tr>
<tr><td>+</td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td>•</td><td>한개이상의 앞문자</td></tr>
<tr><td>g+</td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td>•</td><td>적어도 한개의 g</td></tr>
<tr><td>g\+</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td>우와 같다</td></tr>
<tr><td>?</td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td>•</td><td>령 혹은 한개의 앞문자</td></tr>
<tr><td>g?</td><td></td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td>•</td><td>령 혹은 한개의 g</td></tr>
<tr><td>g\?</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td></td><td>우와 같다</td></tr>
<tr><td>GIF|JPEG</td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td>•</td><td>GIF 또는 JPEG</td></tr>
<tr><td>GIF\|JPEG</td><td></td><td>•</td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td>우와 같다</td></tr>
<tr><td>wood(cock|house)</td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td>•</td><td>woodcock 또는 woodhouse</td></tr>
<tr><td>wood\(cock\|house\)</td><td></td><td>•</td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td></td><td>우와 같다</td></tr>
<tr><td>\<pat</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>단어의 시작에 패턴 pat</td></tr>
<tr><td>pat\></td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>단어의 끝에 패턴 pat</td></tr>
</table>

구간 및 꼬리표 붙은 정규식

이것들은 egrep와 awk에서 사용되지 않는 고급한 정규식들이다. gawk와 perl도 구간정규식을 받아들이지만 대괄호의 앞에 \를 놓는다. gawk는 추가적으로 --posix나 --W re-interval선택항목을 사용할것을 요구한다. perl은 (와)의 앞에도 \를 놓는다.

기호	vi	emacs	grep	sed	egrep	awk	perl	Linux				정 합
								vim	grep	sed	gawk	
\{m\}	•		•	•				•	•	•		m개의 앞문자
{m}							•				•	우와 같다
^.\{9\}nobody	•		•	•				•	•	•		행시작부터 9문자뛰여넘은 다음 nobody
^.{9}nobody							•				•	우와 같다
\{m,\}	•		•	•				•	•	•		적어도 m개의 앞문자
{m,}							•				•	우와 같다
\{m,n\}	•		•	•				•	•	•		m파 n사이의 앞문자
{m,n}							•				•	우와 같다
\(exp\)	•	•	•	•				•	•	•		\1, \2, 등 exp까지 꼬리표 달기
(exp)							•					우와 같으나 $1, $2 등도 사용된다

확장문자렬(escape sequence)

기호	vi	emacs	grep	sed	egrep	awk	perl	Linux				정 합
								vim	grep	sed	gawk	
\b		•					•		•	•		단어경계
wood\b		•					•		•	•		woodcock가 아니라 wood
\B		•					•		•	•	•	단어경계가 아닌 곳
wood\B		•					•		•	•	•	wood가 아니라 woodcock
\w		•					•	•	•	•	•	단어문자([a-zA-Z0-9_]와 같은)
\W		•					•	•	•	•	•	비단어문자([^a-zA-Z0-9_]와 같은)
\d							•	•				수자([0-9]와 같은)
\D							•	•				비수자([^0-9]와 같은)
\s							•	•				공백문자(공간, 타브와 같은)
\S							•	•				공백 아닌 문자
\t		•					•	•			•	타브
\n							•				•	행바꾸기
\r							•				•	자리복귀
\f							•				•	페지넘기기
\nnn							•				•	ASCII 8진수 nnn
\014							•				•	ASCII 8진수 14
\xnn							•				•	ASCII 16진수 xnn

POSIX문자모임

기호	vi	emacs	grep	sed	egrep	awk	perl	Linux				정 합
								vim	grep	sed	gawk	
[[:alpha:]]								•	•	•	•	영문자
[[:lower:]]								•	•	•	•	영어소문자
[[:upper:]]								•	•	•	•	영어대문자
[[:digit:]]								•	•	•	•	수자
[[:alnum:]]								•	•	•	•	영문자와 수자
[[:space:]]								•	•	•	•	페지넘기기를 포함하는 공백문자
[[:cntrl:]]									•	•	•	조종문자
[[:blank:]]								•	•	•	•	공백 또는 tab
[[:print:]]								•	•	•	•	인쇄가능한 문자
[[:punct:]]								•	•	•	•	구두점문자(공백, 문자,수자,조종문자아닌)
[[:xdigit:]]								•	•	•	•	16진수자

부록 4. 쉘참고서

이 부록에서는 4가지의 쉘들 즉 Bourne쉘(sh), C쉘(csh), Korn쉘(ksh), bash의 특징을 보여 준다. sh는 표준프로그람작성언어이지만 csh는 확실하게 우월한 리력치환특성때문에 그럭저럭 유지한다. ksh는 대신 지령의 행내편집을 지원하며 선진적이며 쓸모 있는 몇가지 기능을 제공한다. bash는 이 두가지 가장 좋은것, 다시말하여 csh의 대화적인 특성뿐아니라 sh의 프로그람작성언어도 제공한다. 이 기능들의 대부분은 제8, 10, 17, 18, 19장들에서 취급되였으며 여기서는 그것들을 통합, 간략화하여 비교한다.

지령실행방법

지령은 여러가지 방법 즉 순차적으로, 조건적으로 또는 다른 지령으로부터의 입력을 사용하여 실행시킬수 있다. 스크립트들은 같거나 다른 부분쉘에서 실행될수도 있다. 또한 별명이나 기능을 무시할수도 있고 외부지령이나 내부지령을 실행시킬수도 있다. nohup와 command는 sh안에 내장되여 있지 않으며 흔히 외부지령으로서 존재할수도 있다는것을 주의하시오. 여기서 cmd, cmd1, cmd2는 지령들을 의미한다.

<table>
<tr><th>지 령</th><th>쉘</th><th>실 행</th></tr>
<tr><td>cmd1 ; cmd2</td><td>All</td><td>cmd1후에 cmd2를 실행</td></tr>
<tr><td>cmd1 | cmd2</td><td>All</td><td>cmd1의 표준출력으로부터 입력하여 cmd2를 실행</td></tr>
<tr><td>cmd1 && cmd2</td><td>All</td><td>cmd1이 성공하였을 때에만 cmd2를 실행</td></tr>
<tr><td>cmd1 || cmd2</td><td>All</td><td>cmd1이 실패하였을 때에만 cmd2를 실행</td></tr>
<tr><td>cmd1 `cmd2`</td><td>All</td><td>cmd2의 표준출력으로부터 입력을 받아 cmd1을 실행</td></tr>
<tr><td>cmd1 $ (cmd2)</td><td>ksh, bash</td><td>우와 같다</td></tr>
<tr><td>(cmd1; cmd2)</td><td>All</td><td>보조쉘안에서 먼저 cmd1, 그다음에 cmd2를 실행</td></tr>
<tr><td>{ cmd1; cmd2; }</td><td>sh, ksh, bash</td><td>현재의 쉘안에서 먼저 cmd1, 그다음에 cmd2를 실행</td></tr>
<tr><td>cmd &</td><td>All</td><td>배경에서 cmd를 실행(csh에서 가입탈퇴가 허용된다)</td></tr>
<tr><td>nohup cmd &</td><td>ksh, csh, bash</td><td>배경에서 cmd를 실행(가입탈퇴가 허용된다)</td></tr>
<tr><td>. script</td><td>sh, ksh, bash</td><td>현재의 쉘에서 script를 실행(bash는 source도 사용한다)</td></tr>
<tr><td>source script</td><td>csh, bash</td><td>현재의 쉘에서 script를 실행</td></tr>
<tr><td>\cmd</td><td>All</td><td>외부지령 또는 같은 이름의 별명을 무시한 내부지령을 실행</td></tr>
<tr><td>command cmd</td><td>ksh, csh, bash</td><td>외부지령 또는 같은 이름의 함수를 무시한 내부지령을 실행</td></tr>
</table>

방향절환(redirection)

쉘은 입력, 출력, 오유를 표현하는 3개의 흐름을 사용한다. 사용자는 파일들이 덧쓰기되는것을 막을수 있으며 출력 및 오유흐름들을 병합하여 그것들을 총괄적으로 관리할수도 있다.

h, ksh, bash	csh	의 미
> file	> file	출력을 file로 보낸다
>> file	>> file	출력을 file에 추가한다
< file	< file	file로부터 입력을 얻는다
<< mark	<< mark	문자렬 mark까지의 행들로부터 입력을 얻는다

h, ksh, bash	csh	의 미
set -o noclobber	set noclobber	> 및 >>로 파일을 덧쓰기하지 않는다(sh에서는 무효)
set +o noclobber	unset noclobber	noclobber설정을 반전시킨다(sh에서는 무효)
>| file	>! file	noclobber가 설정되였다고 해도 출력을 file에로 보낸다(sh에서는 무효)
2> file	—	오유통보문을 file에로 보낸다
> file1 2> file2	—	출력을 file1에, 오유를 file2에 보낸다
> file 2>&1	>& file	출력과 오유를 둘 다 file에로 보낸다
2> file 1>&2	>& file	우와 같다
>| file 2>&1	>&! file	noclobber가 설정되였다고 해도 출력과 오유를 file에로 보낸다(sh에서는 무효)
>&n	—	출력을 file서술자 n에로 보낸다

파일이름메타문자

파일이름메타문자들은 2개의 묶음으로 나누인다. 첫번째 묶음에 속하는것들은 보통 모든 쉘에서 공통이며 하나이상의 파일이름을 인수로 사용하는 지령들에서 사용된다. 그것들은 또한 문자렬을 비교하는 case조건문과 현재의 등록부안에서 파일이름들을 비교하는 for순환에 의해서도 사용될수 있다. 파일이름의 시작에 있는 점이 명백히 비교되여야 한다는데 주의를 돌리시오.

sh, ksh, bash	csh	의 미
*	*	임의의 개수(0도 포함)의 문자들
?	?	한 문자
.???*	.???*	점과 그뒤에 적어도 3개의 문자로 시작하는 파일들
[ijk]	[ijk]	i, j, k중의 한 문자(a-z,0-9와 같은 범위들이 허용된다)
[!ijk]	—	i, j, k가 아닌 한 문자(범위가 허용된다)
*.[!oc]	—	C프로그람과 목적파일을 제외하고 모든 확장자를 가진 파일

두번째 묶음의 특성들은 sh에 적용되지 않으며 일부는 csh에 적용되지 않는다. 만일 bash를 사용하고 있다면 패턴의 시작에 부정연산자 !를 사용하기전에 shopt ?s extglob설정을 할 필요가 있다.

ksh, bash	csh	의 미
{p1, p2, p3}	{p1, p2, p3}	파일 p1,p2,p3을 정합한다
calc.{sh,pl,awk}	calc.{sh,pl,awk}	calc.sh, calc.pl, calc.awk를 정합한다
!(pat)	—	pat를 제외한 모든 파일을 정합한다
!(p1|p2|p3)	—	p1, p2, p3을 제외한 모든 파일을 정합한다
!(*.GIF|*.JPEG)	—	.GIF나 .JPEG로 끝나는 파일을 제외한 모든 파일을 정합한다
~	~	현재사용자의 홈등록부를 정합한다
vi ~/.alias	vi ~/.alias	홈등록부에 위치한 .alias를 편집한다
cd ~user	cd ~user	user의 홈등록부로 절환한다
~-	—	이전 등록부
vi ~-/.profile	—	이전에 방문했던 등록부안에 위치한 .profile을 편집한다
cd ~-	—	이전의 등록부로 절환한다
cd -	—	우와 같다

변수 및 배렬에 대한 처리

UNIX변수들은 형을 가지지 않는다. 이것이 수값을 포함하는 변수들이 문자렬과 수로서 취급될수 있는 리유이다. 이 부분은 산수연산기능, 문자렬조작, 배렬과 특수한 파라메터의 특성을 소개한다. 이 특성의 일부는 ksh의 ksh93판본에 적용된다. 여기서 var는 변수를 의미한다.

변수의 초기화

변수를 두가지형 즉 국부변수와 전역변수(환경변수)로 나눌수 있다. csh는 이 변수들을 처리하기 위한 여러가지 명령문들을 사용하며 같은 이름을 가지는 국부 및 전역변수들을 둘 다 관리할수 있다.

sh, ksh, bash	csh	의 미
var=value	set var=value	국부변수 var에 value를 할당한다
var=value	setenv var value	환경변수 var에 value를 할당한다
export var	setenv var value	var의 value를 모든 보조쉘에 통과시킨다
export	setenv	외부변수/환경변수들을 현시한다
unset var	unset var	국부변수 var의 설정을 취소한다
unset var	unsetenv var	환경변수 var의 설정을 취소한다
read var	set var = $<	건반입력을 var로 읽어 들인다

옹근수산수연산

sh를 제외한모든 쉘들은 옹근수산수연산을 수행하기 위하여 +, -, *, /, %연산자를 사용한다. 또한 증가 및 감소연산자(++ 와 --)도 사용한다. sh사용자들은 옹근수연산과 일부 문자렬처리기능을 수행하기 위하여서는 expr에 의존해야 한다. POSIX는 let보다 오히려 (( ))의 사용을 지정하지만 마지막례제는 POSIX가 권고하지 않는 형식인 $를 사용하지 않는다. 이 형식은 오직 bash의 최종판본과 ksh93에서만 동작한다.

sh, ksh, bash	csh	의 미
let z=x+y	@ z = $x + $y	x와 y의 합으로 변수 z를 할당한다
let x=x+1	@ x++	x를 1만큼 증가시킨다
z=$((x+y))	@ z = $x + $y	x와 y의 합으로 변수z를 할당한다
z=$((x/y))	@ z = $x / $y	$x / $y의 상으로 변수z를 할당한다
((x++))	@ x++	x의 1증가 (bash,ksh93)

문자렬처리

이것은 2개의 묶음으로 나누인다. 첫 묶음은 변수가 령 아닌 값으로 설정되였는가에 따라 여러가지로 변수 var의 값을 구한다. 두번째 묶음은 ksh와 bash의 우월한 문자렬처리기능을 표현한다. 이 묶음의 첫 6개 항목은 문자렬로부터 패턴 pat를 교체 또는 삭제한다(pat는 통용기호표현일수 있다).

sh, ksh, bash	csh	값평가
${var:+pat}	—	var가 령 아닌 값을 가진다면 pat
${var:-pat}	—	var가 할당되지 않았거나 령이면 pat
${var:=pat}	—	우와 같으나 var를 pat로 설정한다.
${var:?pat}	—	var가 설정되였으면 $var, 아니면 pat를 현시하고 스크립트를 중단한다.
—	${?var}	var가 할당되지 않았으면 0, 아니면 1

ksh, bash	값평가
${var#pat}	시작에서 pat와 맞는 가장 짧은 토막을 삭제한후 var의 나머지토막
${var##pat}	우와 같지만 가장 긴 토막을 삭제한후 var의 나머지토막
${var%pat}	끝에서 pat와 맞는 가장 짧은 토막을 삭제한후 var의 나머지토막
${var%%pat}	우와 같으나 가장 긴 토막을 삭제한후 var의 나머지토막
${var/pat}	pat의 첫짝을 제거한후의 var
${var/s1/s2}	처음으로 발생하는 문자렬 s1을 s2로 교체한후에 var (bash, ksh93)
${#var}	var의 길이(bash, ksh93)
${var:x:y}	var에서 x위치로부터 y길이만큼의 부분문자렬(bash, ksh93)

배렬처리

배렬의 첫 첨자는 ksh와 bash에서는 0, csh에서는 1이다. ksh와 bash에서는 개별적인 배렬원소들을 독립적으로 설정할수 있는 반면에 csh에서는 첨자를 인식함이 없이 원소들의 범위 또는 마지막배렬원소에 접근할수 있게 한다.

ksh, bash	csh	의 미
var=(val1 val2 …)	set var=(val1 val2 …)	배렬 var에 값 val1, val2 등을 설정한다
unset var	unset var	var를 제거한다
var[n]=value	—	n번째 원소에 value를 할당한다
${var[n]}	$var[n]	n번째 원소
${var[*]}	$var[*]	모든 원소
${var[@]}	$var	우와 같다
${#var[@]}	$#var	원소의 개수
—	$var[$#var]	마지막원소
—	$var[m-n]	m부터 n까지의 원소(m- 와 -n이 허용된다)

특수한 파라메터들과 변수들

쉘은 자동적으로 설정되는 개수의 특수한 파라메터들을 사용한다. 쉘스크립트의 지령행인수들은 스크립트내부에 있는 이러한 일부 파라메터들에로 넘어 간다. csh를 제외한 다른 쉘들은 set지령을 사용하여 그것들의 값을 설정할수도 있다. 다른 파라메터들은 프로쎄스에 관계된다. csh는 다른 쉘들보다 더 직관적인 형식으로 마지막지령행인수를 접근한다.

sh, ksh, bash	csh	의 미
$1, $2,…	$argv[1], $argv[2],…	1, 2 등의 번호가 붙은 지령행인수
$*	$* or $argv[*]	모든 지령행인수들
$#	$#argv	지령행인수의 개수
—	$argv[$#argv]	마지막지령행인수; 다른 쉘들은 var='eval echo\\$$#'를 사용할 필요가 있다.
$0	$0	실행된 지령의 개수
"$@"	—	인용부호로 막힌 여러 단어인수를 한개의 인수로 취급한다
$?	$status	마지막지령의 완료값
$$	$$	현재쉘의 PID
$!	—	마지막배경일감의 PID
—	$<	건반으로부터 읽어 들인 입력

지령리력

sh에는 리력이 없으며 ksh의 특성은 csh나 bash에 의하여 좀 제한된다. 그렇지만 ksh는 vi나 emacs와 류사한 편리성을 사용하여 지령행의 행내편집을 가능하게 한다. 여기서 bash와 csh의 특성은 사실상 동일하지만 bash는 행내편집도 제공하므로 ksh와 csh의 가장 좋은 기능을 제공한다. 여기서 stg, stg1, stg2는 문자렬을 의미한다.

사건의 반복

	ksh	의 미
history n	history -n	최근의 지령들이나 n개의 지령들을 현시한다
!!	r	마지막지령을 반복한다
!n	r n	사건번호 n을 반복한다
!n:p	—	사건 n을 실행하지 않고 현시한다
!-n	r -n	n번 앞의 사건을 반복한다
!-2	r -2	이전 지령의 바로 앞지령을 반복한다
!com	r com	com으로 시작하는 마지막지령을 반복한다
!?stg?	—	문자렬 stg를 가진 마지막지령을 반복한다
!?mtime?:p	—	mtime을 포함하는 마지막지령을 반복한다

이전 지령행의 변경

csh, bash	ksh	의 미
^stg1^stg2	r str1=stg2	stg1을 stg2로 교체한후에 이전 지령을 실행시킨다
!com:s/stg1/stg2	r com stg1=stg2	처음으로 나타나는 stg1을 stg2로 교체한후에 com으로 시작되는 마지막지령을 실행시킨다
!com:gs/stg1/stg2	—	우와 같으나 stg1을 전반적으로 stg2로 교체한후에 실행시킨다
!! arg	r arg	지령행에 인수 arg를 추가하여 이전 지령을 실행한다
!cut | sort	r cut | sort	마지막 cut지령을 실행시킨후에 그것을 sort에 관흐름으로 련결한다
cmd !$	cmd $	이전 지령의 마지막인수를 가진 cmd를 실행시킨다($는 bash에서도 사용된다)

이전 지령의 개별적인수들을 사용

다목적의 지령 및 인수확장기능은 이전 지령의 임의의 인수를 가지고 새로운 지령 또는 이전 지령을 실행시킬수 있게 한다. 이전 지령의 인수는 더우기 지령으로서도 실행될수 있다. 인수들은 일반적인 :n기능을 사용하여 확장된다(여기서 n은 하나의 수 또는 범위이다). :^(또는 :1)과 :$는 첫번째 인수와 마지막인수를 참고하지만 :0은 그 지령만을 골라 낸다. 이 기능은 ksh에서는 쓸수 없으며 csh와 bash에서만 유효하다.

csh, bash	의 미
cmd !*	이전 지령의 모든 인수를 가지고 cmd를 실행한다
cmd !$	이전 지령의 마지막인수를 가지고 cmd를 실행한다
cmd $_	우와 같다(bash에서만 유효)
cmd !:n	이전 지령의 n번째 인수를 가지고 cmd를 실행한다
gzip !:4	이전 지령의 4번째 인수를 가지고 gzip를 실행한다
cmd !:$	이전 지령의 마지막인수를 가지고 cmd를 실행한다
cmd !:m-n	이전 지령의 m번째부터 n번째까지를 가지고 cmd를 실행한다

csh, bash	의 미
compress !ls:3-$	마지막 ls지령의 첫 두개의 인수를 제외한 모든 인수들을 가지고 compress를 실행한다
cmd !:m-	이전 지령의 마지막인수를 제외한 m번째이상의 모든 인수를 가지고 cmd를 실행한다
cmd !m:n	사건 m의 n번째 인수를 가지고 cmd를 실행한다
!m1:0 !m2:n	사건 m2의 n번째 인수를 가지고 사건 m2의 지령을 실행한다
!!:n	이전 지령의 n번째 인수를 지령으로서 실행한다

경로이름을 잘라내기

이전 지령의 어떤 인수가 파일경로이름이라면 csh와 bash는 그 경로이름의 여러 구성요소들을 떼여낼수 있게 한다. 확장자도 기본파일이름으로부터 분리되여 별개로 떨어 져 나올수 있다. csh는 변수들에 대해서도 이 기능을 사용할수 있게 하지만 bash에서는 오직 이전 지령의 인수들로써만 작업한다.

csh, bash	의 미
cmd !$:h	이전 지령의 마지막인수의 머리부를 가지고 cmd를 실행한다
cmd !$:t	이전 지령의 마지막인수의 꼬리부(파일이름)를 가지고 cmd를 실행한다
cp !$!$:t.txt	.txt가 붙은 이전 지령의 마지막인수를 복사하기 위하여 cp를 실행한다
cmd !$:r	이전 지령의 마지막인수의 뿌리를 가지고 cmd를 실행한다
echo !$:e	이전 지령의 마지막인수의 확장자(점없이)를 가지고 echo를 실행한다
echo $var:h	변수 var에 저장된 경로이름의 머리부를 현시하기 위하여 echo를 실행한다(csh에서만)

직렬지령편집

ksh와 bash는 이전 지령이 화면에 현시되는것을 가능하게 한다. 이때 내장된 vi형 또는 emacs형편집기능을 사용하여 그것들을 편집할수 있다. 지령 set -o vi 또는 set -o emacs를 실행시킴으로써 한번에 오직 한가지 방식만을 동작시킬수 있다. 이러한 방식들에서 사용할수 있는 기본적인 지령들을 표 17-4에 보여 주었다.

환경

쉘의 작업환경은 시작파일의 등록들과 변수들, 설정된 방식에 의하여 결정된다. ksh와 bash는 모든 쉘들중에서 가장 좋은 작업환경을 간단히 제공한다. 이 쉘들이 사용하는 시작스크립트들은 표 17-1의 마지막 3개의 등록에 보여 주었다.

환경변수/내장변수

변수들은 습관상 대문자로 표현된다. 그렇지만 csh는 흔히 대문자와 소문자로 같은 변수를 지정한다. 자체의 사용을 위해서는 소문자변수를 사용하지만 외부프로그람들은 대문자변수를 사용한다. 여기서 보여 준 변수들중 4개는 사실 배렬이다. 때때로 어느 한 변수의 설정을 변화시키면 다른것에 영향이 미치지만 항상 그런것은 아니다. ksh와 bash는 많은 추가적인 변수들을 사용한다.

sh, ksh, bash	csh	의 미
CDPATH	cdpath=(dir1 dir2 …)	cd dir를 사용할 때 보이는 등록부들의 목록
HOME	home	사용자의 홈등록부
IFS	—	내부마당구분문자(set에 의하여 사용된다)
LOGNAME	user	사용자의 가입이름
MAIL	mail=(n files)	사용자의 우편함파일(csh에서는 한개이상의 파일)
MAILCHECK	mail=(n files)	우편함파일이 얼마나 자주 검사되는가(n초)

sh, ksh, bash	csh	의 미
MAILPATH	—	하나이상의 두점으로 구분된 우편함파일; 현시된 통보문을 포함할수도 있다
PATH	path=(dir1 dir2 …)	지령이 실행될 때 검색되는 등록부들의 목록
PS1	prompt	초기의 프롬프트문자렬
PS2	—	두번째 프롬프트문자렬(csh에서는 보통 ?)
SHELL	shell	사용자가입쉘과 vi나 mail과 같은 프로그람들에 관한 탈퇴쉘
TERM	term	말단의 형태
USER	user	사용자의 가입이름(bash)
BASH_ENV	—	보조쉘들이 읽어 들이는 환경파일(bash)
EDITOR	—	ksh에서 직렬지령편집에 사용되는 편집기(VISUAL후에 읽혀 진다)
ENV	—	보조쉘들이 읽어 들이는 환경파일(ksh)
HISTFILE	—	리력파일(csh는 .history를 사용한다)
HISTFILESIZE	savehist	리력파일에 보관되는 지령의 수(ksh는 HISTSIZE를 사용한다)
HISTSIZE	history	기억기에 보관되는 지령의 수(ksh에서는 리력파일안에)
LINENO	—	스크립트나 함수에서 행번호
OLDPWD	—	이전의 등록부
PPID	—	현재쉘의 PID
PWD	cwd	현재의 등록부
REPLY	—	인수없이 사용될 때 read에 의하여 사용되는 변수
VISUAL	—	ksh에서 직렬지령편집에 사용되는 편집기(EDITOR를 재정의하며 bash에서는 사용되지 않는다)

쉘방식

ksh와 bash에서 방식은 열쇠단어를 인수로 하여 set -o를 사용한것으로써 설정한다(례: set -o vi). csh는 set를 단순히 열쇠단어와 함께 사용한다(=없이, 례: set filec). 이러한 열쇠단어들을 아래에 보여주었다. 설정은 set +o keyword(ksh, bash) 또는 unset keyword(csh)를 사용하여 제거한다. ksh와 bash에서 이 설정들의 대부분은 한문자선택항목을 가진 set를 사용하여 동작시킬수도 있다.

ksh, bash	csh	의 미
emacs	—	직렬지령편집기를 emacs로 설정한다
vi	—	직렬지령편집기를 vi로 설정한다
ignoreeof	ignoreeof	가입탈퇴에 관하여 eof문자가 무시된다. 오직 exit만이 허락된다
noclobber	noclobber	파일이 >와 >>에 의하여 덧쓰기되는것을 막는다
—	filec	파일이름완성기능을 동작시킨다
noglob	noglob	통용기호를 확장하지 않고 글자그대로 취급한다
nolog	—	리력파일에 기능정의를 저장하지 않는다
notify	notify	일감완성을 즉시 통지하며 다음프롬프트에서는 통지하지 않는다(ksh는 제외)
verbose	—	매 스크립트행을 실행할 때마다 현시한다
xtrace	—	지령행이 실행될 때 앞에 +를 붙여 현시한다(set -x와 같다)

비교검사

쉘들은 문자렬과 옹근수의 비교, 대부분의 파일속성의 검사를 위하여 폭 넓은 자원을 제공한다. 그것들은 보통 if, while, until문자의 협력밑에서 test문 또는 그의 동의어 []와 함께 사용된다. 그것들을 어떻게 사용하고 있을것인가를 아래에 보여 주었다.

```
while [ $x ?lt 10 ] ; do
if ( -e .profile ) then
```

옹근수검사

sh, ksh, bash	csh	의 미
-eq	==	같기
-ne	!=	같지 않기
-gt	>	크기
-ge	>=	크거나 같기
-lt	<	작기
-le	<=	작거나 같기

문자렬검사

마지막 4개의 실례가 [[ ]]연산자들의 사용법을 명백하게 보여 준다. 그것들은 문자렬비교를 위한 통용기호의 사용과 여러 조건들을 평가하기 위한 조건연산자 &&와 ||의 사용을 허락한다. 이 형식들은 Korn쉘의 ksh93판본과 bash의 마지막판본에서 동작한다(SuSE Linux 6.4에 제공된다).

sh, ksh, bash	csh	참(1)으로 평가될 조건
stg	stg	문자렬 stg가 할당되여 있고 null이 아닐 때
-n stg	—	stg가 null문자렬이 아닐 때
-z stg	—	stg가 null문자렬일 때
stg1 = stg2	stg1 == stg2	stg1이 stg2과 같을 때
stg1 != stg2	stg1 != stg2	stg1이 stg2과 같지 않을 때
[[stg == exp]]	stg =~ exp	stg=exp일 때, 여기서 exp는 통용기호패턴일수 있다(sh는 제외)
[[stg != exp]]	stg !~ exp	stg가 통용기호표현 exp와 맞지 않을 때(sh는 제외)
[[stg1 < stg2]]	—	ASCII순서맞추기렬에서 stg1이 stg2보다 앞설 때(sh는 제외)
[[stg1 > stg2]]	—	ASCII순서맞추기렬에서 stg1이 stg2보다 뒤설 때(sh는 제외)

파일검사

sh, ksh, bash	csh	참(1)으로 평가될 조건
-e file	-e file	file이 존재할 때(sh는 제외)
-f file	-f file	file이 보통파일일 때
-d file	-d file	file이 등록부일 때
-b file	—	file이 블로크특성일 때
-c file	—	file이 문자특성일 때
-r file	-r file	file이 읽기가능할 때
-w file	-w file	file이 쓰기가능할 때
-x file	-x file	file이 실행가능할 때

sh, ksh, bash	csh	참(1)으로 평가될 조건
-O file	-o file	file이 사용자에게 소유된것일 때(sh는 제외)
-s file	! -z file	file이 령보다 큰 크기를 가질 때
-L file	—	file이 기호련결일 때(ksh, bash)
-u file	—	file에서 SUID비트가 설정되여 있을 때
-k file	—	file에서 점착비트가 설정되여 있을 때
file1 -nt file2	—	file1이 file2보다 새로운것일 때(ksh, bash)
file1 -ot file2	—	file1이 file2보다 더 낡은것일 때(ksh, bash)
file1 -ef file2	—	file1이 file2에 련결되였을 때(ksh, bash)

내부지령

이 마지막부분은 쉘들의 내부지령을 표현한다. 여러개의 선택항목들이 |로 구분되여 나타난다. select, function, typeset와 같은 일부 구조들은 POSIX의 부분이 아니므로 생략하였다. 그렇지만 이 구조들에 대한 선택은 유효하다. cmd, var, exp, stg는 자기의 일반적인 의미를 가진다.

지령 fg, bg, kill, stop, wait는 jobid를 사용하는데 여기서 jobid는 %기호와 그뒤에 놓이는 일감번호(%n) 또는 문자렬 s로 시작되는 지령(%s), 문자렬 s를 포함하는 지령(%?s), 현재의 일감(%%)이다.

지 령	쉘	의 미
. file	sh,ksh,bash	현재의 쉘에서 파일을 실행시킨다
[ ]	sh,ksh,bash	test문을 위한 동의어
[[ ]]	ksh,bash	우와 같으나 보다 정교한 검사를 허락한다
alias name	ksh,csh,bash	모든 별명정의를 현시하거나 name이 지정되였다면 name의 별명을 현시한다
alias name=cmd	ksh,bash	별명 name을 cmd로서 정의한다
alias name cmd	csh	우와 같다
bg jobids	ksh,csh,bash	현재의 일감 또는 jobid를 배경으로 옮긴다
break n	All	현재의 또는 n번째의 for, while, until의 순환에서 탈퇴한다
case string in pat1) commands1 ;; … patn) commandsn ;; esac	sh,ksh,bash	string이 패턴 patn과 맞으면 commandsn을 실행시킨다. patn은 통용기호표현일수도 있다. 한개의 선택항목에 들어 있는 여러개의 패턴은 |로 구분된다. 선택항목 *는 이전에 대응되지 않은 모든것과 대응된다.
cd	All	홈등록부로 이동한다
cd dir	All	등록부 dir로 이동한다
cd -	ksh,bash	이전 등록부로 이동한다
cd ~/user	ksh,csh,bash	user의 홈등록부로 이동한다
command cmd	ksh,bash	별명이나 함수를 제외한 cmd를 실행시킨다
continue n	All	현재의 또는 n번째의 for, while, until순환의 꼭대기로부터 실행을 시작한다
echo ops stg	All	통보문 stg를 현시한다. csh는 탈퇴절차를 지원하지 않으며 bash는 -e선택항목을 요구한다
eval cmd	All	cmd의 값을 두번 구한다. 변수가 특수한 문자들을 포함할 때 요구된다
exec cmd	All	현재의 쉘을 cmd로 교체한다

지령	쉘	의 미
exec n> file	sh,ksh,bash	파일서술자 n을 file과 련계시킨다
export	sh,ksh,bash	모든 반출된 변수들을 현시한다
export var	sh,ksh,bash	$var의 값을 부분쉘에로 통과시킨다
exit n	All	완료값 0 또는 n(n이 지정된 경우)으로 현재쉘을 끝마친다
fg jobid	ksh,csh,bash	현재의 일감 또는 jobid를 전경으로 옮긴다
for var in list; do 지령들 done	sh,ksh,bash	var를 list로부터 얻은 매값으로 할당하고 do와 done사이의 지령들을 실행시킨다. break와 continue를 참고하시오
foreach var (list) 지령들 end	csh	우와 같으나 foreach와 end사이의 지령들을 실행시킨다
functions	ksh	모든 함수정의를 현시한다

function_name() {
지령들
}
	sh,ksh,bash	지령묶음들에 의해 함수 function_name을 정의한다. 위치파라메터 $1, $2 등을 받아 들이며 선택적으로 return문과 함께 옹근수를 돌려 준다

goto stg	csh	뒤에 두점이 붙은 문자렬 stg로 시작되는 행다음의 지령을 실행한다
history –n	ksh,bash	당면한 사건 또는 마지막 n개의 사건(n이 지정된 경우)의 목록을 현시한다
history n	csh	우와 같다

if 조건 ; then
지령들
선택항목들
endif
	sh,ksh,bash	조건문이 참이면 지령들을 실행한다. 선택항목은 elif 및 else와 함께 교체된 조건을 위한 검사를 포함한다. 18.8을 보시오

if (조건) then
지령들
선택항목들
endif
	csh	우와 같으나 선택항목들은 else if와 else문을 사용한다. 부록 1을 보시오

jobs -l	ksh,csh,bash	모든 일감을(-l이 지정되였다면 PID와 함께) 현시한다
kill -n jobids\pids	ksh,csh,bash	jobids 또는 pids를 가진 하나이상의 프로쎄스를 제거한다. -9는 제거를 확인한다.
let var=exp	ksh,bash	exp의 계산결과를 var에 할당한다. exp는 연산자 +, -, *, /, %를 사용한다
logout	csh	가입쉘을 완료한다
nice +n|-n cmd	All	초기값으로 더 낮은 우선권과 함께 cmd를 실행시키거나 n이 지정되였으면 n단위만큼 우선권을 변화시킨다
nohup cmd	ksh,csh	체계에서 탈퇴하였을 때에도 배경에서 cmd를 실행시킨다
notify	csh	배경일감의 완성을 즉시 통지한다
onintr -|label	csh	새치기신호 2를 조작한다 스크립트는 label로 분기하거나 신호(-)를 무시한다
printf fmt values	ksh,bash	C형의 fmt형식문자렬을 사용하여 형식화된 values를 현시한다
pwd	All	현재의 등록부를 현시한다
repeat n cmd	csh	cmd를 n번 실행한다
read	ksh,bash	스크립트에 입력되는 건반입력을 변수 REPLY에 읽어 들인다
read var	sh,ksh,bash	우와 같으나 입력이 var에 할당된다
set	All	모든 변수를 현시한다(csh에서는 국부변수만)

지령	쉘	의 미
set -o mode	ksh,bash	쉘방식을 설정한다
set exp	sh,ksh,bash	위치파라메터 $1, $2 등을 exp안의 단어들에 설정한다
set var	csh	쉘방식을 설정한다
set var = $<	csh	스크립트에 입력되는 건반입력을 var에로 읽어 들인다
set var = value	csh	국부변수 var에 value를 할당한다
setenv	csh	모든 환경변수들을 현시한다
setenv var value	csh	환경변수 var에 value를 할당한다
shift	All	위치파라메터들을 왼쪽으로 민다. 즉 $2가 $1로 된다
shift n	sh,ksh,bash	우와 같으나 n자리만큼 왼쪽으로 민다
shift var	csh	배렬 var의 원소들을 왼쪽으로 민다
source file	csh,bash	현재의 쉘에서 파일을 실행시킨다
stop jobids	ksh,csh,bash	jobids를 가진 배경일감들을 중지시킨다
suspend	ksh,csh,bash	현재의 전경일감을 중지시킨다([Ctrl-z]와 같다)
switch (string) case pat1: 지령1 breaksw …… case patn: 지령n breaksw endsw	csh	string이 패턴 patn과 정합되면 지령 n을 실행시킨다. switch는 아래로 내려 가면서 breaksw를 만날 때까지 지령들을 실행시킨다. 마지막선택항목(patn)은 default로 교체될수 있으며 이전에 대응되지 않던 모든것이 해당된다
test	sh,ksh,bash	[]를 위한 동의어
trap cmds sigs	sh,ksh,bash	신호처리기. 신호 sigs를 수신할 때 cmds를 실행한다. cmds로서의 공백은 스크립트를 면제시킨다. 신호 0은 현재의 쉘에서 탈퇴할 때 cmds를 실행한다
type cmd	All	cmd가 외부지령, 내부지령, 별명, 함수인가를 밝힌다
ulimit ops	All	자원한계를 설정하거나 현시한다
umask nnn	All	파일만들기마스크를 현시하거나 마스크를 8진표시 nnn으로 설정한다
unalias name	ksh,csh,bash	별명정의 name을 제거한다
unset name	All	변수나 배렬 name의 설정을 해제한다
unset -f name	sh,ksh,bash	함수 name의 설정을 해제한다
until 조건 ; do 지령들 done	sh,ksh,bash	조건이 참이 될 때까지 do와 done사이의 지령들을 실행한다
wait	All	모든 배경일감들이 완성될 때까지 실행을 잠시 중지한다
wait pids	sh,ksh,bash	PID pids를 가진 일감들이 완성될 때까지 실행을 잠시 중지한다
while 조건 ; do 지령들 done	sh,ksh,bash	조건이 참일 동안 do와 done사이의 지령들을 실행한다
while (조건) 지령들 end	csh	조건이 참일 동안 while과 end사이의 지령들을 실행한다

부록 5. 지령일람표

지 령	조 작	페 지
. (점)	보조쉘을 생성함이 없이 쉘스크립트를 실행시킨다	478
accept	인쇄대기렬에 일감을 받아 들이는것을 허락한다	659
alias	지령렬을 생략한다	458
apropos	열쇠단어를 포함하는 지령을 현시한다	48
at	1회실행을 위한 일감의 시간표를 작성한다	285
awk	행안의 개별적인 마당들을 조작한다	426
batch	체계부하가 허용될 때 1회일감의 시간표를 작성한다	286
bc	대화형으로 산수연산을 진행한다	63
bg	일감을 배경으로 돌린다	284
cal	달력 또는 년력을 현시한다	61
calendar	시간일지(engagement diary)를 관리한다	62
cancel	인쇄작업을 취소한다	169
cat	파일들을 련결시킨다	166
cat	파일내용을 현시한다	
cat >	파일을 만든다	166
cd	현재의 등록부를 홈등록부로 바꾼다	158
cd dirname	현재의 등록부를 바꾼다	157
chat	인터네트에 접속하기 위하여 ISP에 회선으로 련결한다	681
chgrp	파일의 그룹소유권을 바꾼다	194
chmod	파일의 허가권을 바꾼다	185
chown	파일의 소유권을 바꾼다	193
chsh	관리자의 참가없이 가입쉘을 바꾼다	445
cmp	두 파일사이의 차이점을 현시한다(문자목록으로서)	245
comm	두 파일에 공통인 행 또는 한 파일에 유일한 행을 현시한다	247
compress	파일을 압축한다(.Z로)	172
cp	파일을 복사한다	163
cp -r	등록부나무를 복사한다	164
cpio	표준입력으로부터 얻은 목록을 가지고 파일들을 대피시킨다	640
cpio	여벌(backup)매체로부터 파일들을 회복한다	640
cron	반복실행을 위한 일감의 시간표를 작성한다	287
crontab	cron을 위한 특성파일을 생성한다	288
cut	파일로부터 렬이나 마당을 잘라 낸다	250
date	체계날자를 설정한다(체계관리자만)	627
date	체계날자를 현시한다	61
dd	플로피디스크나 테프매체를 복사한다	639
df	빈 디스크공간을 현시한다	170
diff	두 파일사이의 차이점을 현시한다(sed형지령으로써)	246
dip	인터네트에 접속하기 위하여 ISP에 회선으로 련결한다	680

지 령	조 작	페 지
disable	인쇄기를 금지시킨다	
dos2unix	DOS로부터 UNIX로 파일을 변화시킨다	260
doscat	DOS디스크안의 파일내용을 현시한다	640
doscp	DOS디스크에, DOS디스크로부터 파일을 복사한다	640
dosdir	DOS디스크의 파일목록을 현시한다	639
dosformat	DOS플로피디스크를 초기화한다	639
du	디스크공간소비정형을 알아 본다	171, 649
echo	통보문을 현시한다	218
echo "\007"	경보음을 낸다	218
egrep	여러 패턴중의 하나이상을 포함하는 행들을 현시한다	407
elm	우편을 보낸다	339
emacs	파일내용을 현시한다	
emacs	파일을 편집한다	107
enable	인쇄기를 허락한다	659
eval	배렬형으로 번호화된 변수들을 생성한다	252
exec	흐름을 파일에 련결시킨다	558
exec	현재의 쉘을 다른 프로그람으로 교체한다	558
exit	가입대화를 완료한다	14
exit	쉘스크립트를 완료한다	501
export	변수값을 보조쉘에 보낸다	448
expr	비대화형으로 산수연산을 진행한다	511
expr	문자렬길이를 현시한다	
expr	더 큰 문자렬로부터 추출된 부분문자렬을 현시한다	513
expr	더 큰 문자렬안에서 부분문자렬의 위치를 현시한다	513
fdformat	UNIX플로피디스크를 형식화한다	638
fdisk	디스크구획을 만들거나 수정한다	
fetchmail	POP나 IMAP봉사기로부터 우편을 꺼낸다	708
fg	일감을 전경으로 돌린다	284
fgrep	여러 패턴중의 하나이상을 포함하는 행들을 현시한다	407
file	파일형에 따르는 파일분류를 현시한다	166
find	이름 또는 최종변경시간, 접근시간으로 파일을 찾는다	201, 650
finger	가입하지 않을 때에도 사용자정보를 현시한다	299
format	디스크구획을 만들거나 수정한다	616
fsck	파일체계의 완전성을 검사한다(체계사용자만)	620
ftp	기계들사이에서 파일을 복사한다	303
grep	패턴을 포함하는 행들을 현시한다	397
grep -c	패턴을 포함하는 행의 수를 계수한다	398
grep -l	패턴을 포함하는 파일목록을 현시한다	401
grep -v	패턴을 포함하지 않는 행들을 현시한다	400
grep -v	빈 행들을 제거한다	400
groupadd	사용자그룹을 추가한다(체계관리자만)	629
groupdel	사용자그룹을 제거한다(체계관리자만)	628
groupmod	사용자그룹을 수정한다(체계관리자만)	629
gunzip	압축된 .gz파일을 푼다	172

지 령	조 작	페 지
gzip	파일을 압축한다(.gz 로)	172
head	파일의 머리부를 현시한다	248
history	지령리력을 현시한다	462
hostname	국부주콤퓨터의 이름을 현시한다	294
hostname	주콤퓨터이름을 설정한다(체계관리자만)	670
ifconfig	망대면부를 구성한다(체계사용자만)	670
info	지령문서를 여러 준위로 현시한다	45
init	체계의 실행준위를 설정한다(체계관리자만)	634
init	체계를 끈다(체계사용자만)	636
irc	망에서 여러 사용자들사이의 대화(chat)를 진행한다	373
ispell	문서의 맞춤법검사를 진행한다	262
kill	프로쎄스를 완료한다	282
kill $!	마지막배경일감을 완료한다	283
less	한번에 한 페지씩 파일내용을 현시한다	239
let	쉘을 리용하여 산수연산을 진행한다	539
ln	파일에 대한 련결을 생성한다	
ln -s	파일에 대한 기호련결을 생성한다	199
lock	체계에서 탈퇴함이 없이 말단을 차단한다	57
logout	가입대화를 완료한다	14
lp	파일을 인쇄한다	168
lpadmin	인쇄기를 관리한다(체계관리자만)	657
lpc	인쇄기를 관리한다(체계관리자만)	662
lpc disable	인쇄대기렬에 일감을 받아 들이는것을 금지시킨다	662
lpc enable	인쇄대기렬에 일감을 받아 들이는것을 허락한다	662
lpc start	인쇄기를 허락한다	662
lpc stop	인쇄기를 금지시킨다	662
lpq	인쇄대기렬을 현시한다	169
lpr	파일을 인쇄한다	169
lprm	인쇄작업을 취소한다	169, 660
lpstat	인쇄대기렬을 현시한다	658
ls	파일목록을 현시한다	177
ls -l	파일속성을 현시한다	181
ls -l | grep "^d"	등록부목록을 현시한다	406
lynx	WWW페지들을 비도형방식으로 현시한다	378
mail	우편을 보낸다	338
mailq	우편대기렬을 현시한다	702
man	지령문서를 현시한다	60
mcopy	DOS디스크에, DOS디스크로부터 파일을 복사한다	640
mdir	DOS디스크의 파일목록을 현시한다	639
mesg	talk를 위하여 말단을 허락 또는 금지시킨다	298
mformat	DOS플로피디스크를 초기화한다	639
minicom	인터네트에 접속하기 위하여 ISP에 회선으로 련결한다	679
mkfs	파일체계를 생성한다	615
more	한번에 한 페지씩 파일내용을 현시한다	238

지 령	조 작	페 지
mount	파일체계를 태운다	617
mtype	DOS디스크안의 파일내용을 현시한다	639
mv	파일을 다른 등록부에로 옮긴다	165
mv	파일 또는 등록부들의 이름을 고친다	165
netscape	우편을 보낸다	348
netscape	WWW페지들을 도형방식으로 현시한다	378
netscape	망새소식을 처리한다	370
netstat	망상태를 현시한다	674
nice	일감의 우선권을 증가시킨다(체계관리자만)	280
nice	일감의 우선권을 감소시킨다	280
nl	빈 행을 제외하고 행들에 번호를 붙인다	259
nohup	배경에서 지령을 실행하고 가입에서 탈퇴한다	279
nslookup	IP주소와 FQDN사이의 변환을 진행한다	699
nslookup	령역에 관한 우편봉사기들을 현시한다	699
nslookup	령역에 관한 이름봉사기들을 현시한다	699
od	문자의 8진값을 현시한다	242
passwd	자기의 통과암호를 바꾼다	53
passwd usname	임의의 사용자의 통과암호를 바꾼다(체계관리자만)	627
paste	두개의 파일을 그대로 련결시킨다	251
perl	행안의 개별적인 마당들을 조작한다	567
perl	HTML형식의 자료를 처리한다	567
pine	우편을 보낸다	342
ping	주콤퓨터의 접속을 시험한다	672
pr	머리부와 페지수를 가진 파일내용을 현시한다	243
pr -d -t	두줄공간으로 행들을 현시한다	244
pr -k	k렬로 파일내용을 현시한다	244
pr -n -t	빈 행을 포함하여 행들에 번호를 붙인다	244
ps	프로쎄스속성을 현시한다	274
ps f	프로쎄스계통(process ancestry)을 현시한다	277
ps -f	프로쎄스계통(process ancestry)을 현시한다	274
pwd	현재의 등록부를 검사한다	155
rcp	인증없이 기계들사이에서 파일을 복사한다	309
read	자료를 쉘스크립트에 대화식으로 입력한다	492
reject	인쇄대기렬에 일감을 받아 들이는것을 금지시킨다	659
rlogin	통과암호를 사용하지 않고 원격기계에 가입한다	302
rm	파일을 제거한다	165
rm -r	본문으로부터 행바꾸기문자를 제거한다	165
rmdir	빈 등록부를 제거한다	162
route	경로조종표의 경로들을 관리한다(체계관리자만)	673
route -n	경로조종표를 현시한다(체계관리자만)	681
rsh	가입없이 원격기계상에서 지령을 실행한다	310
runlevel	체계실행준위를 현시한다	634

지 령	조 작	페 지
script	대화를 기록한다	59
sed	빈 행들을 제거한다	410
sed	파일로부터 여러개의 토막들을 현시한다	410
sed	한 패턴을 다른것으로 치환한다	416
sed	파일의 여러 토막들을 여러 파일로 써넣는다	415
set	위치파라메터에 값을 할당한다	534
set -x	쉘스크립트의 오유수정작업(debug)을 진행한다	561
setenv	변수값을 보조쉘에 보낸다	448
shift	위치파라메터들을 더 낮은 번호로 민다	536
shutdown	체계를 끈다(체계사용자만)	636
sleep	스크립트안에서 지령실행을 지연시킨다	515
sort	ASCII순서로 행들을 현시한다	252
sort -f	대소문자를 무시하고 정돈된 행들을 현시한다	255
sort -n	수자적인 순서로 행들을 현시한다	253
sort -u	중복된 행들을 제거한다	254
source	보조쉘을 생성함이 없이 쉘스크립트를 실행시킨다	478
spell	문서의 맞춤법검사를 진행한다	261
startx	X Window체계를 기동한다	317
stty	말단특성을 설정한다	57
su	보통등록자리로부터 상급사용자를 현시한다	602
sync	완충기내용들을 디스크에 써넣는다(체계관리자만)	621
tail	파일의 끝을 현시한다	249
tail -f	파일의 증대를 감시한다	249
tail -r	반대순서로 행들을 현시한다	249
talk	망에서 두 사용자사이의 대화를 진행한다	297
tar	여벌(backup)매체로부터 파일들을 회복한다	644
tar	지령행에 지정된 파일을 대피시킨다	644
tee	하나의 흐름을 두개로 가른다	227
telnet	인증후에 원격기계에 가입한다	300
time	프로그람을 실행시키는데 걸리는 시간을 현시한다	289
tin	망새소식을 처리한다	391
top	기억기와 교체구역(swap)의 빈 용량을 현시한다	278
top	체계기억기의 리용상태를 현시한다	278
touch	파일의 최종변경시간 또는 접근시간을 바꾼다	196
tput clear	화면을 지운다	60
tput cup	화면상에 위치유표를 현시한다	60
tr	본문의 대소문자를 바꾼다	256
tr	한 문자를 다른것으로 치환한다	
tr -d	본문에서 행바꾸기문자를 제거한다	256
tr -s	여러개의 공간을 하나의 공간으로 압축한다	257
trap	쉘스크립트로부터 신호를 처리한다	562
tty	현재말단의 장치이름을 현시한다	56

지 령	조 작	페 지
type	지령의 형(외부, 내부 또는 별명)을 현시한다	35
ulimit	최대파일크기를 현시한다	628
ulimit	최대파일크기를 설정한다 (체계관리자만)	628
umask	파일의 기정허가권을 현시한다	191
umask	초기의 파일허가권을 설정한다 (체계관리자만)	191
umount	파일체계를 내리운다	618
uname	조작체계이름을 현시한다	60
uname -r	조작체계의 판번호를 현시한다	41
uncompress	압축된 .Z파일을 푼다	172
uniq	정돈된 파일로부터 중복된 행들을 제거한다	258
uniq -d	반복된 행들을 현시한다	259
uniq -u	오직 한번만 발생하는 행들을 현시한다	259
unix2dos	UNIX로부터 DOS로 파일을 변환한다	260
unzip	압축된 .zip파일을 푼다	172
useradd	사용자등록자리를 추가한다(체계관리자만)	630
userdel	사용자등록자리를 제거한다(체계관리자만)	631
usermod	사용자등록자리를 수정한다(체계관리자만)	631
users	사용자이름만 현시한다	55
vacation	수신자가 떨어 져 있을 때 모든 송신자에 대한 통보문을 보낸다	355
vi	파일을 만든다	167
vi	파일을 편집한다	67
w	체계리용상태와 사용자정보를 현시한다	55
wall	모든 사용자들에게 주소를 지정한다 (체계관리자만)	627
wc	행과 단어, 문자의 수를 계수한다	240
whatis	지령소개를 한행으로 현시한다	48
who	사용자들과 그의 동작을 현시한다	55
who -r	체계실행준위를 현시한다	635
xargs	표준입력으로부터 받은 인수와 함께 지령을 실행시킨다	651
xcalc	대화형으로 산수연산을 진행한다	329
xclipboard	X안에서 여러본문단락을 복사한다	328
xhost	X봉사기에로의 접근을 조종한다	297
xinit	X Window체계를 기동한다	317
xkill	X창문을 완료한다	330
xload	원격기계의 체계부하를 현시한다	329
xrdb	X자원파일을 다시 읽어 들인다	331
xterm	X Window안에서 쉘로부터 지령을 실행시킨다	325
zcat	압축된 파일내용을 현시한다 (.Z 혹은 .gz)	172
zip	여러 파일을 한개의 파일로 압축한다 (.zip로)	173

부록 6. ASCII문자표

이 부록은 ASCII문자모임의 첫 128개 문자들의 값을 10진수, 16진수, 8진수로 보여 주고 있다. 8진값들은 UNIX지령 awk, echo, perl, tr에 의하여 사용되며 한편 od는 문자들을 8진수로 현시한다. 이 지령들의 대부분은 일부 문자들에 대하여 비고란에서 보여 준것과 같이 \x형식의 확장문자렬(escape sequence)도 사용한다. awk와 perl도 16진값을 사용한다.

문자	10진수	16진수	8진수	비고
(null)	0	00	000	Null
[Ctrl-a]	1	01	001	
[Ctrl-b]	2	02	002	
[Ctrl-c]	3	03	003	
[Ctrl-d]	4	04	004	
[Ctrl-e]	5	05	005	
[Ctrl-f]	6	06	006	
[Ctrl-g]	7	07	007	삑소리문자(\a)
[Ctrl-h]	8	08	010	공백(\b)
[Ctrl-i]	9	09	011	타브(\t)
[Ctrl-j]	10	0A	012	행바꾸기(\n)
[Ctrl-k]	11	0B	013	수직타브(\v)
[Ctrl-l]	12	0C	014	페지넘기기(\f)
[Ctrl-m]	13	0D	015	자리복귀(\r)
[Ctrl-n]	14	0E	016	
[Ctrl-o]	15	0F	017	
[Ctrl-p]	16	10	020	
[Ctrl-q]	17	11	021	
[Ctrl-r]	18	12	022	
[Ctrl-s]	19	13	023	
[Ctrl-t]	20	14	024	
[Ctrl-u]	21	15	025	
[Ctrl-v]	22	16	026	
[Ctrl-w]	23	17	027	
[Ctrl-x]	24	18	030	
[Ctrl-y]	25	19	031	
[Ctrl-z]	26	1A	032	
[Ctrl-[]	27	1B	033	
[Ctrl-\]	28	1C	034	
[Ctrl-]]	29	1D	035	
[Ctrl-^]	30	1E	036	
[Ctrl-_]	31	1F	037	
(space)	32	20	040	공백
!	33	21	041	감탄부호
"	34	22	042	겹인용부호
#	35	23	043	파운드기호
$	36	24	044	딸라기호
%	37	25	045	퍼센트기호

문자	10진수	16진수	8진수	비고
&	38	26	046	앰퍼샌드기호
'	39	27	047	외인용부호
(	40	28	050	왼쪽괄호
)	41	29	051	오른쪽괄호
*	42	2A	052	별표
+	43	2B	053	더하기기호
,	44	2C	054	반점
-	45	2D	055	이음표
.	46	2E	056	점
/	47	2F	057	사선기호
0	48	30	060	
1	49	31	061	
2	50	32	062	
3	51	33	063	
4	52	34	064	
5	53	35	065	
6	54	36	066	
7	55	37	067	
8	56	38	070	
9	57	39	071	
:	58	3A	072	두점
;	59	3B	073	반두점
<	60	3C	074	작기부호
=	61	3D	075	같기부호
>	62	3E	076	크기부호
?	63	3F	077	물음표
@	64	40	100	애트기호
A	65	41	101	
B	66	42	102	
C	67	43	103	
D	68	44	104	
E	69	45	105	
F	70	46	106	
G	71	47	107	
H	72	48	110	
I	73	49	111	
J	74	4A	112	
K	75	4B	113	
L	76	4C	114	
M	77	4D	115	
N	78	4E	116	
O	79	4F	117	
P	80	50	120	
Q	81	51	121	
R	82	52	122	
S	83	53	123	

문자	10진수	16진수	8진수	비고
T	84	54	124	
U	85	55	125	
V	86	56	126	
W	87	57	127	
X	88	58	130	
Y	89	59	131	
Z	90	5A	132	
[	91	5B	133	왼쪽꺾쇠괄호
\	92	5C	134	역사선기호
]	93	5D	135	오른쪽꺾쇠괄호
^	94	5E	136	탈자기호
_	95	5F	137	밑선
`	96	60	140	역인용부호
a	97	61	141	
b	98	62	142	
c	99	63	143	
d	100	64	144	
e	101	65	145	
f	102	66	146	
g	103	67	147	
h	104	68	150	
i	105	69	151	
j	106	6A	152	
k	107	6B	153	
l	108	6C	154	
m	109	6D	155	
n	110	6E	156	
o	111	6F	157	
p	112	70	160	
q	113	71	161	
r	114	72	162	
s	115	73	163	
t	116	74	164	
u	117	75	165	
v	118	76	166	
w	119	77	167	
X	120	78	170	
Y	121	79	171	
Z	122	7A	172	
{	123	7B	173	왼쪽대괄호
|	124	7C	174	막대기호 혹은 관련결기호
}	125	7D	175	오른쪽대괄호
~	126	7E	176	물결표
	127	7F	177	삭제

부록 7. 용어해설

가상조종탁(virtual console)
하나의 UNIX콤퓨터로부터 여러개의 화면과 가입들을 리용하는 체계. [Alt]와 기능건을 사용하여 새 화면을 펼친다.

가상주콤퓨터(virtual host)
Apache(Apache)를 포함한 많은 httpd봉사기들에서 유용한 기능으로서 한대의 기계상에서 여러개의 싸이트를 운영한다. 매 가상주콤퓨터는 자기의 봉사기이름과 cgi-bin등록부, 문서뿌리를 가질수 있다. 가상주콤퓨터는 기본주콤퓨터와 같은 IP주소를 가지거나 다른 IP주소를 가질수 있다.

가입(login)
사용자가 가입이름을 입력할 때 getty프로그람을 공급하는 처리. 가입이 성공하면 쉘스크립트를 실행시킨다.

가입등록부(login directory)
홈등록부와 같다.

가입이름(login name)
사용자ID와 같다.

감돌기(wraparound)
파일의 다른 끝으로부터 패턴탐색을 재개하기 위하여 vi와 emacs편집기가 제공하는 기능. 따라서 탐색개시순간의 유표위치에는 상관없이 전체 파일이 탐색된다. vi에서는 최종행방식지령 :set nowraparound를 사용하여 금지시킬수 있다.

건맺기(key binding)
emacs지령을 건입력렬에 련계시키는것. 유효한 건렬이 눌릴 때 emacs는 내부적으로 그 건에 결부된 지령을 실행시킨다.

겹쳐놓기(overlay)
프로쎄스의 생성에서 요구되는 최종단계. 원래의 프로그람코드를 새로운 코드로 교체하기 위하여 exec(쉘명령문이나 체계호출)가 사용하는 원리이다.

경련결(hard link)
⇨ 련결

경로기(router)
관문(gateway)과 같다.

경로조종(routing)
다른 망에 속하는 파케트들을 관문이나 경로기로 보내는것. 또한 경로조종표는 명시적인 경로들로 저장될수 없는 모든 파케트들에 기정경로를 제공한다.

경로이름(pathname)
/으로 구분된 하나이상의 파일이름들의 렬. 마지막파일이름을 제외한 모든 파일이름들은 등록부이여야 한다. ⇨ 상대경로이름, 절대경로이름

공백(whitespace)
IFS변수에 의하여 설정되는 공간(space), 타브(tab), 행바꾸기(newline)들의 련속적인 렬. 지령행인수들을 분석하기 위하여 많은 려파기들과 쉘이 구분문자(delimiter)로서 사용한다. set문도 자기의 인수들을 위치파라메터에 할당하기 위하여 이것을 사용한다.

공통관문대면부(Common Gateway Interface: CGI)
Web봉사기가 외부응용프로그람에 양식자료를 넘겨 주기 위하여 제공하는 대면부. 응용프로그람이 그 자료를 처리하여 그 결과를 의뢰기열람프로그람에 돌려 보낸다. perl은 CGI에서 사용되는 가장 공통적인 언어이다.

공통탁상환경(Common Desktop Environment: CDE)
현재 거의 모든 UNIX제품들에 적용된 X Window체계하에서의 탁상전반에 대한 표준화된 보기(look)와 느끼기(feel). 응용프로그람을 시작할수 있는 정면판(Front Panel)과 파일관리기(File Manager), 다중탁상화면(multiple desktops)의 사용을 허락하는 작업공간스위치(Workspace Switch)로 특징 지어 진다. CDE가 사용하는 창문관리기 dtwm은 Motif에 기초하고 있다.

교갑화(encapsulation)
TCP/IP망에서 통신규약탄창에서 내려 갈 때 파케트에 정보가 추가되는 처리. 송신측에서는 매층이 자기의 머리부와 꼬리부를 추가하며 그것은 수신측에서 벗겨 진다.

교체(swapping)
현재 비능동인 프로쎄스들을 주기억으로부터 디스크의 교체구역에로 옮기는 처리. 또한 실행준비가 되였을 때 이 프로쎄스들을 림시구역으로부터 주기억으로 전송하는것도 의미한다. 교체구역은 분리된 파일체계안에 일련의 련속적인 블로크들로 구성된다.

구간정규식(interval regular expression: IRE)
\{ 와 \}로 둘러 막힌 하나 또는 한쌍의 수(반점으로 구분)를 사용하는 정규식. 두개의 수는

그앞에 놓인 한 문자가 발생할수 있는 회수의 최소, 최대값을 가리킨다. grep, sed, perl지령들이 이것을 사용하며 perl에서는 \ 가 사용되지 않는다.

구획(partition)
파일체계를 보유하기 위한 하드디스크의 별도의 구역. 한 구획의 자료는 다른 구획으로 쪼개질수 없다. 하나의 디스크상에 4~8개의 구획이 있을수 있다. Linux는 여러개의 론리구획을 보유할수 있는 또 하나의 확장구획을 지원한다.

구역(region)
vim과 emacs에서 일부 동작(action)을 배치하기 위하여 표식된 편집구역. 그 령역은 일부 외부지령들을 통하여 복사, 삭제, 이동, 려파될수도 있다. vim은 그 구역을 강조하지만 emacs는 그렇게 하지 않는다.

그룹(group)
파일허가권을 처리할 때 chmod지령이 리해하는 사용자의 한가지 부류. 둘이상의 사용자들이 그룹에 속할수도 있고 한 묶음의 파일허가권이 이 부류와 련관될수 있다. 수값적인 표현은 /etc/passwd와 /etc/group안에 저장되며 이름은 그다음에 저장된다.

그룹식별자(group-id:GUID)
사용자의 그룹이름 또는 번호. 사용자등록자리를 만들 때 체계관리자가 분배한다. 이름과 수값적 표현은 /etc/group안에 저장되며 수값은 /etc/passwd에서도 유효하다.

기다리기(wait)
새끼프로쎄스가 실행되고 있는 동안 어미프로쎄스가 휴식하는것을 가리키는 용어. 보통 어미프로쎄스는 새끼프로쎄스의 완료를 기다린다. 또한 이 이름을 가진 쉘내장지령도 있다.

기동블로크(boot block)
모든 파일체계에 있는 특정한 구역으로서 기본파일체계에서는 이 블로크에 기동절차(boot procedure)와 구획표가 포함되며 다른 파일체계들에서는 이 구역이 비여 있다.

기본번호(major number)
장치에 접근하기 위하여 요구되는 장치구동프로그람을 가리키는 장치파일목록의 파라메터중의 하나. 류사한 장치들은 같은 기본번호를 가진다. ⇨ 보조번호

기정경로(default route)
국부망으로 향하지 않은 모든 파케트들에 기정탈퇴경로를 제공하기 위한 경로조종과 관련된 용어. PPP는 인터네트로 향한 모든 파케트들을 자기의 대면부에 이끌어 가기 위하여 그것을 선택항목(defaultroute)으로서 사용한다. 모든 주콤퓨터도 류사한 기정경로를 제공하는 경로조종표를 관리한다.

기호련결(symbolic link)
파일이나 등록부의 위치를 지정하는 파일. 경련결(hard link)들과는 달리 기호련결은 파일체계들을 넘어 가면서 파일들을 련결할수 있다. 등록부들을 련결하는데도 리용될수 있다. ⇨ 련결

귀환주소(loopback address)
모든 기계에 유효한 가상적인 망대면부. 127.0.0.1로 주소화된 통보문을 《귀환》시키며 모든 망봉사들이 단독주콤퓨터상에서 실행되게 한다.

관문(gateway)
여러개의 망에 속해 있으면서 적어도 2개의 망대면부기판을 가지는 콤퓨터. 관문은 파케트들에 한 망으로부터 다른 망에로의 경로를 제공한다. ⇨ 경로기

관흐름(pipeline)
한 지령의 출력이 다른 지령의 입력으로 되도록 한개이상의 |기호를 리용한 2개이상의 지령들의 입력렬. ⇨ 표준입력, 표준출력

권한 없는 사용자(nonprivileged user)
체계관리자의 권한을 가지지 않는 보통의 사용자.

닉명ftp(anonymous ftp)
사용자들이 가입이름으로서는 닉명을, 통과암호로서는 전자우편주소를 사용하여 접근하는 대중적인 ftp싸이트. 대체로 내리적재가능한 쏘프트웨어들이 이러한 싸이트들에서 관리된다. 파일의 올리적재(upload)는 허용되지 않는다.

내리우기(unmounting)
파일체계를 주파일체계로부터 떼여 내는 처리. 지령 umount가 이 처리를 수행하며 체계를 끌 때 체계에 의하여 사용된다. 체계관리자는 파일체계의 완전성검사를 진행하기전에 그 파일체계를 내리운다.

내부지령(internal command)
vi, emacs, more, mail, sed지령과 같은 많은 UNIX도구들과 쉘의 부분지령으로 주어 진 이름.

다른 사용자(other)
파일허가권을 처리할 때 chmod지령에 의하여 해석되는 사용자의 한 부류. 파일의 소유자도 아니고 그 그룹소유자에도 속하지 않는 사용자들이 이 부류에 속한다. 한조의 파일허가권이 여기에 관계된다. ⇨ 소유자, 그룹

다목적인터네트우편확장(Multipurpose Internet Mail Extensions: MIME)
인터네트상에서 2진파일을 부호화하는데 리용되는 규격. 하나의 우편통보문안에 여러가지 자료형식을 부호화하는데도 유용하다. 다매체첨부물을 우편으로 보내는데 리용된다. MIME형식은 /etc/mime.types안에 저장된다.

단순우편전송규약(Simple Mail Transfer Protocol: SMTP)
전자우편자료를 인터네트상에서 전송하는데 쓰이는 TCP/IP통신규약. SMTP는 상대측의 SMTP봉사기와 통신하며 통보문을 직접 배포한다. SMTP를 구현한 가장 공통적인 프로그람이 sendmail이다. ⇨ 우편전송대행체

단어(word)
공백을 포함하지 않는 문자들의 련속적인 렬. wc가 단어들의 단위를 계수하는 선택항목을 지정하는 한편 일부 GNU려파기(부록 3)들은 그것을 맞추기 위하여 특수한 기호들을 사용한다. 쉘은 인용된 문자렬을 그가 포함할수 있는 실제적인 단어의 개수에는 상관없이 한개의 단어로서 리해한다.

담화(chat)
⇨ 인터네트중계담화

도전맞잡기인증규약(Challenge Handshake Authentication Protocol:CHAP)
공유비밀(shared secret)과 도전문자렬(challenge string)의 개념을 리용하는 인증체계. CHAP는 /etc/chap-secrets안에 저장된 공유비밀과 도전문자렬로부터 계산을 진행하여 량측의 계산이 맞을 때에만 접근을 허락한다. CHAP는 아주 안전하며 ISP에서 보편적으로 리용되고 있다. ⇨ 통과암호인증규약

도형사용자대면부(graphical user interface: GUI)
보기와 느끼기를 조종하는 X Window체계의 구성요소. GUI의 보기특성은 특수한 의뢰기 즉 창문관리기(window manager)에 의하여 결정된다.

돌림값(return value)
완료상태(exit status)와 같다.

동기(sync)
파일체계와 프로쎄스들과의 접속에 쓰이는 용어. 핵심부는 주기억으로부터 상위블로크와 색인마디자료를 디스크에 써넣기 위하여 sync지령을 사용한다. 쓰기조작을 완성하는데는 2회의 sync이면 충분하다.

동작(action)
주소로 지정한 본문에 작용하는 sed, awk, perl 지령의 구성요소. 보통 sed를 위한 동작을 표현하는데는 한 문자를 사용하지만 awk와 perl의 경우에는 완전한 프로그람일수 있다. 때때로 내부지령이라고도 한다.

등록부파일(directory file)
UNIX체계의 3가지 형태의 파일중에서 자료를 포함하지 않고 다른 파일들과 보조등록부들의 이름을 넣어 두는 한가지 형태. 매 파일에 대한 색인마디번호와 파일이름을 포함한다. 등록부파일에 대한 써넣기는 오직 핵심부만이 가능하다.

대면부프로그람(interface program)
인쇄를 위하여 자료를 려파하는 쉘스크립트. 대면부프로그람은 자료를 양식화하고 인쇄에 필요한 코드들을 제공하며 파일을 인쇄하는 외부응용프로그람을 불러 낸다.

데몬(daemon)
사용자가 특별히 요구하지 않고 주기적으로 실행되는 처리. cron, init, pppd, inetd, sendmail, lpsched는 체계를 가동시키는 중요한 데몬들이다.

려파기(filter)
표준입력으로 문자렬을 얻고 그 내용을 처리하며 표준출력에로 류사한 문자렬을 발생시키는 UNIX지령. 쉘의 입출력절환 및 관련결기능이 이 지령들에 사용될수 있다. 려파기는 그의 자료의 원천 및 목적지를 알지 못한다.

련결[1](concatenation)
두개이상의 실체들의 결합. cat지령과 쉘변수들과의 접속에 사용되는 용어이다.

련결[2](link)
색인마디안에 저장되는 파일속성으로서 파일이 한개이상의 이름으로 참조되도록 한다. 지령은 ln이라는 이름을 가진다. 경련결(hard link)과 같다. ⇨ 기호련결

렬거(listing)
매 파일의 7가지 속성을 보여 주는 ls -l지령으로 얻어 지는 출력.

령역(domain)
완전지정주콤퓨터이름(fully qualified hostname:FQDN)들의 부분으로서 여러 주콤퓨터들이 사용하는 공동문자렬. 인터네트상에서 웃준위령역(top-level domain)들의 실례를 들면 com, edu, org 등이다. ⇨ 지역

령역이름체계(Domain Name System: DNS)
령역과 지역의 개념을 리용하여 망내의 주콤퓨

터의 이름을 유일적으로 서술하는 TCP/IP망에서 리용하는 봉사. 주콤퓨터이름과 IP주소사이의 변환을 수행하는 기능도 제공한다. 주소넘기기(mapping)를 포함하는 자료기지가 해당기관의 필연적인 위탁으로 대규모망에 배포된다.

론리블로크(logical block)
디스크I/O조작에 리용되는 바이트의 묶음을 한정하는데 쓰이는 단위. 보통 1024byte이며 디스크상에서는 1byte를 포함하는 파일도 하나의 론리블로크와 2개의 물리블로크를 차지하게 된다. ⇨ 물리블로크

리력(history)
C쉘, Korn쉘, bash에서 이전 지령들을 저장하고 재호출하고 실행시키는 기능. 이 쉘들에서 지령을 그 이름으로 특징 짓는다.

림시중지(suspend)
일감을 림시적으로 정지시키는 처리. 그 일감은 후에 제거될수도 있고 배경으로 옮겨 지거나 전경에서 다시 시작될수도 있다. 이 기능은 C쉘, Korn쉘, bash에서 유효하다.

마디(node)
임의의 망가입자-장치를 가리키는 용어.

망새소식전송규약 (Network News Transfer Protocol:NNTP)
망새소식이나 새소식그룹을 처리하는데 사용되는 TCP/IP통신규약. 한 새소식봉사기는 또 다른 새소식봉사기로부터 새소식을 가져 온다. Netscape Messenger도 역시 NNTP의뢰기로서 동작하지만 tin과 trn이 공통적인 문자기반의 NNTP의뢰기이다.

망파일체계(Network File System: NFS)
사용자들이 국부등록부상에 원격파일체계의 등록부를 태울수 있게 하는 TCP/IP응용프로그람. 접근권한은 원격체계의 /etc/exports에 의하여 조종된다.

명령(instruction)
주소와 동작(action)의 결합. 주소는 그 작용의 영향을 받는 행들을 지정한다. sed, awk, perl에서 사용된다.

모뎀(modem)
상사신호를 수자신호로, 수자신호를 상사신호로 변환하는 장치(modulator-demodulator). 회선을 통하여 인터네트나 임의의 TCP/IP망에 접속할 때 사용된다.

무한순환(infinite loop)
완료될수 없는 while 또는 until순환. 조종을 순환의 밖으로 절환시키기 위하여 break문(perl에서는 last문)이 사용된다.

문맥주소(context address)
한쌍의 사선(/)으로 닫긴 정규식을 사용하는 sed, awk, perl에서 쓰이는 주소지정형식. 표현을 포함하는 행들은 그 작용의 효과를 받는다.

문서뿌리(document root)
HTML문서들을 저장하는 Web봉사기안의 절대경로이름. Apache구성에서는 지령 DocumentRoot에 의하여 조종된다.

문자(character)
임의의 체계에서 볼수 있는 정보의 최소단위. 건반의 눌림이 한개의 문자를 발생하며 ASCII는 그들중 128개의 묶음을 가진다.

문자형장치(character device)
출력을 문자들의 흐름으로 읽거나 쓰는 말단 또는 인쇄기. 완충기억기는 무시되며 자료가 직접 읽혀 진다. 목록에서 허가권마당의 첫 문자위치에 문자 c로 나타낸다. ⇨ 블로크형장치

물리블로크(physical block)
디스크에 저장된 자료를 한정하기 위하여 몇개의 UNIX지령들이 사용하는 단위. 대부분의 체계들에서 512byte로 정해 진다(Linux에서는 1024). ls, df, du, find는 물리블로크단위로 통보를 내보낸다.

믿을수 있는 주콤퓨터(trusted host)
원격주콤퓨터가 자기의 자원에 접근할수 있도록 믿을수 있는 주콤퓨터. 주콤퓨터가 믿을수 있으면 그 주콤퓨터의 모든 사용자들도 믿을수 있다. 망에서 믿을수 있는 주콤퓨터는 보안의 위험요소로 된다.

매지크(magic)
vi에서 쓰이는 용어로서 정규식에서 사용되는 문자의 특수한 의미를 가리킨다. 최종행방식지령(:set nomagic)을 사용하여 매지크를 해제할수 있다.

메타건(meta key)
emacs지령을 불러 내기 위하여 다른 건들과 결합하여 사용되는 건반의 조종건. PC상에서는 [Esc]나 [Alt]건으로 표현된다.

메타문자(metacharacter)
쉘에 한정된것들을 의미하는 문자들의 묶음에 주어 지는 이름. 쉘은 그 지령을 실행시키기전에 이 문자들에 작용한다. 이 문자들중 일부의 의미는 문자의 앞에 \을 붙임으로 하여 반대로 된다. 그 개념은 또한 문법의 부분으로서 일정한 지령

들에 의해 사용되는 특수한 문자들로 확장된다. ⇨ 통용기호

반복(iteration)
순환명령묶음이 되풀이되는것. 순환고리안의 명령문들은 그 순환지령행안에 지정된 조건식이 참(true)일 동안 반복된다. while, until, for 순환들에 관계되여 쓰이는 용어이다.

반복인자(repeat factor)
지령의 접두사로서 수자를 리용하는 vi편집기, more, less지령에서 유용한 기능. 보통 그 수값의 회수만큼 지령을 반복한다. emacs에서는 수자인수라고 한다.

반전스위치(toggle switch)
직전의 동작의 효과를 뒤집는 지령. emacs는 이러한 스위치로 동작하는 몇개의 지령을 가지고 있다.

방송(broadcast)
기계의 MAC주소를 얻기 위하여 TCP/IP가 망안의 모든 기계들에 보내는 통보문. 방송주소를 결정하기 위해서는 IP주소의 주콤퓨터부분의 모든 비트들을 1로 설정한다.

방식행(mode line)
emacs화면에서 제일 아래에서부터 두번째 행으로서 파일이름, 행번호, 변경상태, 편집기의 방식을 현시하는데 사용된다.

방조응용프로그람(helper application)
열람프로그람이 파일확장자로 표현되는 특정한 파일형식을 처리하기 위하여 불러 내는 외부프로그람. 삽입프로그람(plugin)과 달리 파일들을 분리된 창문에 현시한다. 파일확장자, 내용의 형식, 그를 처리하기 위하여 필요한 외부프로그람은 /etc/mime.types와 /etc/mailcap안에 지정된다.

방향절환(redirection)
쉘에서 지령의 입출력을 재할당하기 위하여 쓰이는 용어. 자료흐름의 기본적인 원천 및 목적지가 디스크파일을 가리키도록 재정의될수 있다.

방화벽(firewall)
외부의 침입으로부터 망내부를 보호하기 위하여 특수한 쏘프트웨어를 실행시키는 기계. 망내부의 주콤퓨터들은 방화벽을 통하여 외부세계에 접속해야 한다. 선택적인 접근을 제공하는 추가적인 쏘프트웨어로서 구성될수 있으며 대리봉사기(proxy server)로서도 동작한다.

변경시간(modification time)
색인마디안에 저장되는 파일의 시간도장의 하나로서 파일의 내용이 최종적으로 변경된 날자와 시간을 표현한다. 목록에 현시되는 속성들중의 하나이다.

별명(alias)
지령렬이나 주콤퓨터이름, 전자우편주소의 또 다른 이름으로 참고하기 위하여 사용되는 용어. C쉘, Korn쉘, bash에서 긴 지령렬을 생략하는데 쓸모가 있다. DNS는 주콤퓨터에 또 다른 이름을 제공하기 위하여 이것을 리용한다. sendmail은 또 다른 주소로 우편을 발송하기 위하여 이 기능을 사용한다.

보조번호(minor munber)
장치의 특정한 특성을 가리키는 장치파일목록의 파라메터들중의 하나. 장치구동프로그람에 보내여 지는 파라메터를 의미하는것으로 해석할수 있다. ⇨ 기본번호

보조쉘(sub-shell)
어미쉘에 의하여 생성된 두번째 쉘. 보통 쉘스크립트나 ( )연산자를 가진 지령들의 묶음을 실행시키기 위하여 요구된다. 보조쉘에 만들어 진 변화는 어미쉘에 영향을 주지 않는다.

보존파일(archive)
한 묶음의 파일들을 하나의 단위 즉 자성매체나 하나의 디스크파일로서 저장하는데 쓰이는 용어. tar와 cpio가 그러한 단위들을 생성한다.

보통파일(ordinary file)
프로그람이나 자료, 본문을 표현하는 UNIX체계의 가장 공통적인 파일. 될수록 많은 자료를 포함하지만 파일의 끝표식이나 임의의 파일속성은 포함하지 않는다. ⇨ 정규파일, 파일

봉사기(server)
⇨ 의뢰기-봉사기구성방식

부분루틴(subroutine)
쉘함수와 같이 perl에서 묶음으로 실행되는 명령문들의 모임. 부분루틴들은 배렬 @_안에 저장된 인수들을 사용한다. perl은 부분루틴을 호출하기 위하여 &기호를 사용한다.

부분망마스크(subnet mask)
TCP/IP망에서 망주소부분이 모두 1로 설정된 주콤퓨터의 IP주소. 이 마스크는 망이 부분망으로 되였는가 아닌가를 결정한다.

부호화(encryption)
문자렬을 무질서하게 발생되는 문자순서로 부호화하는 방법. 체계의 모든 인증된 사용자들의 통과암호를 저장하는데 쓰인다.

분석기(resolver)
TCP/IP응용프로그람에서 리용되는 서고루틴의

묶음으로서 이름봉사기에 령역이름을 IP주소로 분석하기 위한 질문을 보낸다. 파일 /etc/resolv.conf에 의하여 표현된다.

블로크형장치(block device)
출력을 바이트가 아니라 블로크의 단위로 읽고 쓰는 하드디스크나 테프장치, 플로피구동기. 자료읽기는 먼저 완충기억기로부터 시도된다. 목록에서는 허가권마당의 첫 문자위치에 문자 b로써 가리킨다. ⇨ 문자형장치

빈 정규식(empty regular expression)
두개의 사선으로 지정되는 빈(null) 문자렬로서 이것은 작용될 문자렬이 검색된 마지막문자렬과 같다는것을 가리킨다. sed에서 치환을 수행하는데 리용된다.

배경(background)
프로그람(어미프로그람은 그가 완료되기를 기다리지 않는다.)이 실행되는 환경. 지령의 끝에 & 기호가 붙으면 쉘은 그 지령이 배경에서 실행되는것으로 리해한다. 배경일감이 nohup지령이 없이 실행된 경우에는 사용자가 체계에서 탈퇴할 때 완료된다. C쉘과 bash에는 적용되지 못하는 제한이 있다.

사멸(death)
프로쎄스의 완료를 가리키기 위하여 쓰이는 용어. 프로쎄스는 그를 표현하는 지령이 실행을 완성하였을 때 사멸된다.

사용자ID설정방식(set-user-id:SUID)
파일에 할당된 특수한 방식. 그 파일을 실행시키는 사용자는 그 프로그람이 동작중인 동안 그 파일의 소유자의 권한을 획득한다. 사용자들이 직접적으로가 아니라 특수한 지령을 사용하여 중요한 체계파일을 수정하게 하는 UNIX가 사용하는 방식이다. 허가권마당에서 문자 s로 표현된다.

사용자등가성(user equivalence)
원격주콤퓨터상의 같은 등록자리에 대한 사용자접근을 조종하기 위한 r-편의프로그람들의 문맥에서 사용되는 용어. 어떤 사용자가 사용자등가성을 가지고 있다면 그가 통과암호를 사용함이 없이 rlogin을 사용하여 원격주콤퓨터안에 있는 자기의 등록자리로 가입하는것이 허용된다. 사용자등가성은 원격기계상에 있는 $HOME/.rhosts와 /etc/hosts.equiv에 의하여 조종된다.

사용자식별자(user-id:UID)
체계에로의 접근을 얻기 위하여 사용자가 허용하는 이름. 인증된 이름들의 목록이 수값표현과 함께 /etc/passwd에서 관리된다. 가입이름, 사용자이름이라고도 한다.

사용자이름(username)
사용자식별자와 같다.

삽입프로그람(plugin)
열람프로그람이 조작할수 없는 특수한 파일형식을 처리하기 위하여 열람프로그람에 설치되는 작은 프로그람. 방조응용프로그람과 달리 파일을 편집하는데 사용될수 없다.

상급사용자(super user)
체계관리자와 같다.

상대경로이름(relative pathname)
현재등록부와 관련되는 파일의 위치를 정의하는 경로이름. 기호 .과 ..을 사용하여 현재등록부와 어미등록부를 각각 가리킨다. ⇨ 절대경로이름

상위블로크(superblock)
모든 파일체계에서 그의 크기와 상태(빈 블로크와 색인마디들의 상세한 정보와 같은)를 반영하는 특정한 구역. sync지령이 체계의 기억기표를 여기에 규칙적으로 써넣는다.

서명파일(signature file)
사용자의 홈등록부에 있는 .signature파일. 모든 우편통보문과 함께 보내야 할 개인의 세부정보를 입력하는데 쓰인다. 대부분의 우편사용자대행체들이 모든 송신통보문에 대하여 그 파일에 자동적으로 접촉하도록 구성된다.

서표(bookmark)
Web문서에 남겨 두는 보이지 않는 표식. 사용자로 하여금 중간련결을 통하지 않고 직접 그 위치로 이행할수 있게 한다. emacs는 또한 특정한 행위치에로 파일을 직접 불러 내는데도 서표를 사용한다.

선택항목(option)
일반적으로 -로 시작되는 문자렬로서 지령의 본래의 동작을 변화시킨다. 그의 인수들중의 하나를 형성할수도 있다. 일반적으로 여러개의 선택항목들이 하나의 -기호에 결합될수 있다.

소케트(socket)
원천지와 목적지를 각각 가리키는 2개의 포구번호와 2개의 IP주소의 유일적인 결합. 2개의 접속이 같은 소케트를 가질수 있는것은 아니다. ⇨ 포구번호

소형완충기(minibuffer)
emacs화면의 마지막행으로서 사용자가 입력한 지령렬과 체계통보문을 현시하는데 쓰이는 구역. 탐색본문이 여기에 입력된다. 방식행의 뒤에 놓

인다.

소유자(owner)

파일의 내용과 허가권을 결정하는데 완전한 권한을 가지고 있는 그 파일을 생성한 사용자. 파일허가권을 처리할 때 chmod지령은 그를 사용자로 해석한다. 문자렬과 수값적인 표현은 /etc/passwd에 저장된다. ⇨ 그룹, 다른 사용자

수자인수(digit argument)

지령을 여러번 반복시키기 위하여 emacs지령이 사용하는 앞붙이수. vi에서는 반복인자(repeat factor)라고 한다.

스레드(thread)

어떤 통보문에 대응하여 교환되는 통보문들을 묶음화하는데 사용되는 용어. 스레드들은 전자우편과 새소식그룹통보문에서 다 볼수 있다. 많은 우편사용자대행체들과 새소식읽기프로그람(Netscape와 같은)들은 스레드들에 의하여 통보문을 묶고 현시한다.

스팸(spam)

사용자에게 다량적으로 수신되는 불필요한 우편. Netscape와 같은 많은 우편사용자대행체들은 통보문안의 문자렬들을 탐색하여 그것들을 지워버리거나 다른 곳으로 옮겨 버리는 스팸려과기를 제공한다.

시작스크립트(start script)

봉사를 시작하기 위하여 사용되는 S로 시작하는 rc스크립트.

신호(signal)

어떤 사건이 발생하였다는것을 통지하기 위하여 프로쎄스와 통신하는 수단. 신호들은 새치기건을 누르거나 kill지령을 사용하는것에 의하여 발생된다. kill -l지령은 체계에 적용되는 모든 신호들을 보여 준다.

실행준위(run level)

UNIX체계의 여러가지 상태를 가리키는 용어. 실행준위는 init지령의 인수들에 의하여 결정된다. 이 실행준위의 값에 따라 서로 다른 rc스크립트들이 실행된다.

새소식그룹(newsgroup)

UNIX기반의 USENET로부터 기원된 인터네트상의 비직결토론묶음. 통보문은 모든 성원들에게 가닿지 않으며 오직 그것들을 새소식봉사기로부터 내리적재해야 한다. NNTP통신규약을 사용한다. tin, trn과 Netscape Messenger가 보편적인 새소식읽기프로그람이다.

새치기(interrupt)

완료를 목적으로 프로쎄스에 신호를 보내는것. 특정한 건이 이 일감에 할당되며(보통 [Ctrl-c] 또는 [Delete]) stty지령으로 그것을 재할당할수 있다. 신호번호 2를 가진다.

새끼프로쎄스(child process)

어미프로쎄스로부터 생성된 프로쎄스. 생성된 프로쎄스는 환경파라메터의 일부를 어미프로쎄스로부터 물려 받지만 새끼프로쎄스에 만들어 진 환경변화는 어미프로쎄스에 영향을 주지 않는다.

색인마디(inode)

파일의 속성들을 저장하기 위하여 디스크의 특정한 구역에 유지되는 구조. 이 표는 매 파일에 관하여 허가권, 소유권세부정보, 시간도장, 련결의 수를 포함한다. 그러나 파일이름은 포함하지 않는다.

색인마디번호(inode number)

파일에 관한 색인마디를 식별하는 유일번호. 이 번호는 색인마디목록안의 그 색인마디의 위치를 가리킨다. ls의 선택항목 -i로 그것을 현시한다.

생성[1](spawn)

어떤 지령을 실행시키기 위하여 새끼프로쎄스를 만드는것으로서 어미프로쎄스는 그의 완료를 기다린다. 그러나 쉘의 대부분의 내부지령들은 프로쎄스의 새끼치기가 없이 실행된다.

생성[2](birth)

프로쎄스를 만드는것을 가리키는데 리용되는 용어. 프로쎄스는 그것을 표현하는 지령이 호출될 때 만들어 지며 지령의 실행이 완료될 때 제거된다.

생존(keepalive)

영구접속(persistent connection)과 같다.

쉘(shell)

UNIX체계의 지령해석기로서 모든 가입말단들에서 영구적으로 실행된다. 쉘은 사용자요구를 처리하며 핵심부와 작용하여 그 지령을 실행시킨다. 또한 프로그람작성능력도 가지고 있다. 이 책에서는 4가지 형태의 쉘들을 설명하고 있다.

쉘수속(shell procedure)

쉘스크립트와 같다.

쉘스크립트(shell script)

지령들의 한 묶음을 포함하는 보통파일로서 보조쉘에서 해석적인 방법으로 실행된다. 모든 쉘내부지령들과 UNIX외부지령들이 스크립트안에 지정될수 있다. 쉘프로그람 또는 쉘수속이라고도 한다.

쉘프로그람(shell program)

쉘스크립트와 같다.

쉘함수(shell function)
현재의 쉘에서 하나의 묶음으로서 실행되는 명령문들의 묶음. 쉘함수는 파라메터들을 받아 들이며 론리값만을 돌려 줄수 있다. 항상 현재의 쉘에서 실행된다.

자동보관(autosave)
완충기를 주기적으로 별개의 파일에 보관하는 emacs편집기의 기능. 자동보관되는 파일의 이름에는 량쪽에 #가 붙으며 편집기의 recover-file 지령으로 회복할수 있다.

자유목록(free list)
핵심부가 파일의 만들기나 추가를 위해 즉시적으로 해제할수 있는 파일체계의 상위블로크안에서 관리되는 색인마디들의 목록.

자유쏘프트웨어재단(Free Software Foundation)
GNU와 같다.

자원레코드(resource record)
BIND에 의하여 사용되는 자료기지파일의 레코드. A레코드는 FQDN을 IP주소로 변환한다. PTR는 그 반대변환을 수행한다. CNAME은 별명을 제공하는데 리용되며 MX는 그 지역에 관한 우편을 수신하는 봉사기를 제공한다.

잠자기(sleep)
프로쎄스의 림시정지를 가리키는 용어 또는 그것을 수행하는 지령의 이름.

장치구동프로그람(device driver)
모든 장치들을 조작하기 위하여 UNIX핵심부안에 내장된 루틴들의 묶음. 핵심부는 정확한 장치구동프로그람을 호출하여 거기에 일정한 인수들을 보낸다. 장치목록의 기본번호와 보조번호가 이 파라메터들을 가리킨다. ⇨ 기본번호, 보조번호

장치파일(device file)
UNIX체계에서 장치를 표현하는 3가지 형태의 파일중의 하나. 디스크파일과의 호상작용이 실제적으로 물리적장치의 동작으로 이루어 지도록 통신통로를 제공한다.

전경(foreground)
어떤 일감(어미는 그 프로쎄스가 완료되기를 기다린다.)이 실행되는 환경. 보통 전경에서는 하나의 일감만을 실행시킬수 있다.

전송조종규약(Transmission Control Protocol: TCP)
자료의 전송을 담당한 TCP/IP계렬의 가장 중요한 통신규약의 하나. 자료를 토막(segment)들로 쪼개고 상대측에서 그것들을 조립한다. 잃어 버린 토막들은 재전송되므로 전송은 전적으로 믿을수 있다.

절대경로이름(absolute pathname)
파일이 절대적인 방식으로 즉 뿌리로부터 지정되여야 한다는것을 가리키는 /으로 시작되는 경로이름. ⇨ 상대경로이름

점대점규약(Point-to-Point Protocol: PPP)
직렬포구(대체로 모뎀을 통하여)상에서 실행되는 TCP/IP통신규약. 사용자들은 보통 자기들의 기계와 ISP사이에 PPP련결을 리용하여 인터네트에 접속한다. 봉사기와 의뢰기에서 다 실행될수 있는 pppd지령이 PPP를 대표한다.

점착비트(sticky bit)
파일이나 등록부에 할당된 특수한 방식. 보통파일의 실행코드는 일단 그것이 실행되였다면 교체구역(swap area)에 고착된다. 점착비트가 설정된 등록부는 사용자그룹에 의하여 공유되며 여기서 한 사용자는 다른 사용자의 파일을 지우거나 변경시킬수 없다. 목록의 허가권마당에서 문자 t로 나타낸다.

접근시간(access time)
파일이 마지막으로 접근된 날자와 시간을 표현하는 색인마디안에 저장되는 파일의 시간도장(time stamp)들의 하나. 파일을 읽거나 쓰거나 실행시키면 접근된것으로 간주되며 접근시간은 ls -lu지령에 의하여 현시된다.

정규식(regular expression)
일부 특수문자와 보통문자들로 형성되는 모호한 식으로서 한개이상의 문자렬을 대조하기 위하여 지령에 의하여(쉘에 의해서가 아니라) 전개된다. 개념적으로 쉘의 통용기호와 류사하며 행의 어떤 위치에 있는 패턴을 비교하는 기능을 특징 짓기도 한다. 쉘에 의한 해석을 방지하기 위하여 정규식은 항상 인용부호로 둘러 막힌다. ⇨ 메타문자

정규파일(regular file)
보통파일과 같다.

조종지령(control command)
구조체의 조종흐름을 결정하기 위하여 쉘의 지령행 및 awk, perl의 조건문이나 순환에서 쓰이는 지령.

조종파일(control file)
일부 지령들이 자기의 명령문들을 꺼내오는 본문파일. .exrc, /etc/sendmail.cf, /etc/inittab, /etc/resolv.conf가 UNIX체계에서 볼수 있는 전형적인 조종파일들이다.

주소(address)
동작의 영향을 받도록 행들을 지정하는 sed,

awk, perl지령의 구성요소. 특성은 단일행번호나 그것들의 범위, 정규식이나 그 묶음, 두가지의 결합으로 구성된다.

주콤퓨터(host)
망에서 개별적인 IP주소를 가지는 콤퓨터나 장치.

주콤퓨터파일(hosts file)
주콤퓨터이름-IP주소넘기기를 포함하는 파일 /etc/hosts를 가리킨다.

주콤퓨터이름(hostname)
망에서 유일한 주콤퓨터의 이름. 흔히 인터네트상에서 FQDN을 표현하기 위한 일련의 문자렬과 함께 쓰인다. 주콤퓨터이름을 현시하고 설정하는 지령의 이름도 hostname이다.

증가탐색(incremental search)
emacs에서 유용한 빠르고 효과적인 탐색체계. 문자가 입력되자마자 탐색이 시작되며 입력된 문자렬에 일치되는 첫 문자렬로 유표가 즉시 이동한다.

지령(command)
일반적으로 프롬프트에 입력되는 첫 단어. 보통 실행가능한 파일이지만 쉘의 내장명령문(내부지령이라고도 함)들과 일부 다른 지령들(mail, vi 등과 같은)도 포함한다.

지령대입(command substitution)
역인용부호(` `)의 한짝으로 막힌 보조지령. 쉘은 그 지령을 실행시키고 그 지령이 발생시킨 출력이 일어 난 곳에 지령본문을 배치한다.

지령방식(command mode)
vi편집기에서 유용한 3가지 방식중의 하나로서 건눌림이 본문에 작용하는 지령으로 해석되게 한다. 이 방식에서는 건이 눌리우면 화면상에 보이지는 않고 그의 효과만 나타난다.

지령행(command line)
쉘의 프롬프트에 지정되는 지령과 그의 선택항목, 파일이름, 기타 인수들의 완전한 렬. 쉘은 완전한 지령행을 만날 때에만 지령을 실행시킨다.

지역(zone)
별도로 관리되며 자체의 이름봉사기에 의하여 조작되는 령역의 한 부분. 지역을 위한 이름봉사기가 그 지역을 대표한다. ⇨ 령역

지역전송(zonal transfer)
주이름봉사기(master name server)로부터 DNS자료를 종속이름봉사기(slave name server)에로 전송하는 동작. 이 전송은 BIND의 notify문에 의하여 이루어 지기도 하며 두 봉사기가 동조상태를 유지하도록 도와 준다. ⇨ 이름봉사기

직접삽입(inline)
분리된 창문이나 응용프로그람을 사용함이 없이 열람기창문안에서 본문곁에 도형을 배치하는것. Netscape에서 GIF와 JPEG파일들이 직접삽입으로 현시된다.

직접편집(in-place editing)
perl에서 쓰이는 용어로서 파일을 편집하고 그 출력을 입출력절환을 사용함이 없이 같은 파일에 되돌려 쓴다.

질문문자렬(query string)
<이름=값>형식으로 Web봉사기에 양식자료를 보내는 문자렬. 이 문자렬들은 GET방식이 사용될 때에는 QUERY_STRING환경변수들에 유용하지만 POST가 사용될 때에는 표준입력으로서 공급된다.

집선기(hub)
주콤퓨터들의 묶음으로부터 우편을 접수하고 그것들의 배포를 중심적으로 조작하는 중심기계. 흔히 들어 오는 모든 우편을 처리하며 접수한 우편을 주콤퓨터들에 발송할수도 있다. 집선기는 흔히 주콤퓨터이름을 은폐하기 위해서 또는 그 우편이 집선기로부터 출발한것처럼 보이게 하기 위해서 발송자의 주소를 고쳐 쓴다.

재귀(recursion)
지정된 등록부를 내리훑어 그 등록부아래의 모든 보조등록부들에 접근하기 위한 일부 UNIX지령들의 특수한 기능. ls, rm, chmod, chown, chgrp들이 이것을 수행하기 위한 특수한 선택항목을 사용하며 find와 tar는 필수적으로 그 기능을 수행한다.

제거(kill)
프로쎄스에 신호를 보내여 그것을 완료시키는것. UNIX에서는 또한 신호번호와 프로쎄스PID를 사용하여 그 프로쎄스를 제거하는 지령의 이름으로도 리용된다. 일반적으로 일단 어미프로쎄스가 완료되면 모든 새끼프로쎄스들도 제거된다. 현재 거의 모든 쉘들에 내장되여 있는 기능이다.

제거고리(kill ring)
마지막 30개의 삭제 및 복사를 저장하는 emacs의 림시저장구역. 대부분의 삭제조작은 삭제된 자료를 제거고리에 보낸다. 제거고리안에 등록된 것은 회복될수 있다.

제거스크립트(kill script)
봉사를 제거하기 위하여 사용되는 K로 시작하

는 rc스크립트. ⇨ rc스크립트

창문관리기(window manager)
모든 X의뢰기의 보기와 느끼기를 조종하는 특수한 X의뢰기프로그람. 모든 창문주위에 틀을 만들며 창문의 크기변경 및 이동을 가능하게 한다. CDE의 dtwm이 출현하기전까지 mvm이 가장 보편적인 창문관리기프로그람이였다. Linux는 KDE와 GNOME이 출현하기전까지 표준창문관리기로서 fvwm을 사용하였다.

창문부분품(widget)
X Window체계안에서 사용자대면부객체를 정의하는데 사용되는 용어. 차림표들과 흘림띠들, 단추들로 구성된다. Athena와 Motif가 대표적인 형태의 창문부분품묶음들이다.

첨부물(attachment)
전자우편통보문에 속하여 전송되는 파일. 2진파일일수 있으며 우편의뢰기에서 즉시로(inline) 또는 삽입프로그람(plugin)이나 방조(helper)응용프로그람을 사용하여 현시할수 있다.

체계관리자(system administrator)
체계자원의 관리를 담당한 사람. 관리자는 임의의 파일속성을 변경시킬수 있고 임의의 사용자프로쎄스를 제거할수 있다. 관리의무를 수행하기 위하여 특수한 사용자등록자리(보통 뿌리)를 사용한다. 상급사용자라고도 한다.

체계프로쎄스(system process)
사용자가 특별히 요구함이 없이 기동시에 체계내에서 발생되는 프로쎄스. 체계프로쎄스들의 일부를 보면 init, getty, cron, lpsched 등이다.

체계호출(system call)
핵심부안에 정의된 기초루틴으로서 사용자가 UNIX체계안의 모든 복잡한 프로쎄스들을 여전히 알지 못하게 한다. 그러한 루틴들의 몇가지는 파일프로쎄스, 관련결의 생성, 프로쎄스의 발생과 같은 중요한 과제들을 수행한다. 모든 지령들과 프로그람들은 체계호출의 한계내에서 서술된다.

최종행방식(last line mode)
vi편집기에서 유효한 한가지 방식으로서 ex지령을 본문상에서 리용할수 있게 한다. 치환, 여러파일조작, 편집기의 전용화에 필수적인 방식이다. vi에서 두점을 눌러 이 방식을 불러 낸다. ⇨ ex방식

캐쉬(cache)
자료전송속도를 높일 목적밑에 자주 요구되는 정보를 디스크가 아니라 기억기안에 저장하기 위하여 많은 응용프로그람들이 제공하는 기능. TCP/IP망작업 특히 이름봉사기에서 쓰이는 용어이다. Netscape도 최근에 얻은 Web페지를 저장하기 위하여 이 기능을 사용한다.

타브(tab)
공간의 련속적인 모임을 모의하는 한 문자. 특정한 건이나 [Ctrl-i]를 눌러 발생시키며 공백(whitespace)문자들중의 하나로 된다. 렬을 맞추는데 쓸모가 있다.

통과암호(password)
가입시에 모든 사용자들이 사용하는 비밀코드. 이 코드는 말단상에 현시되지 않으며 /etc/shadow안에 은페된 방법으로 저장된다. 류사한 이름인 passwd가 통과암호를 변화시키는 지령을 가리킨다.

통과암호로화(password aging)
통과암호를 변경시키기 위한 조건을 설정하는 체계. 보통 이 변경을 위한 최대, 최소시간이 있다. passwd지령과 함께 선택항목을 사용하여 실현한다.

통과암호인증규약(Password Authentication Protocol: PAP)
PPP가 주콤퓨터들을 인증하기 위하여 사용하는 체계. ISP들은 인터네트에로의 접근을 허락 또는 불허하기 위하여 량측에서 /etc/ppp/pap-secrets안에 저장된 공유비밀의 개념을 사용한다. PAP는 통과암호가 평문으로 망에 보내여 지기때문에 그리 안전하지 못하다. ⇨ 도전맞잡기인증규약

통용기호(wildcard)
한 묶음의 파일이름들을 하나의 표현으로 일치시키기 위하여 쉘이 사용하는 특수한 문자. * 와 ?는 표현들을 구성하는데 사용되는 대표적인 통용기호이다. ⇨ 메타문자

태우기(mounting)
단독의 파일체계를 기본파일체계에 접속하는 처리. 태운후에 그의 뿌리등록부는 태우기가 진행된 파일체계의 등록부로 된다. 또한 기동시에 주파일체계로 모든 단독체계들을 통합하기 위하여 체계가 사용하는 지령의 이름은 mount이다.

파케트(packet)
TCP/IP망에서 자료의 쪼각화단위를 표현하는데 적용되는 용어. 동의어로서 데타그람(datagram)도 리용된다.

파일(file)
정보를 저장하는 용기. 보통의 파일은 자료를 포함한다. 등록부파일은 매 파일들의 색인마디번호와 파일이름을 포함한다. 장치파일에로의 접근은

물리적장치에로의 접근을 실현한다. 흔히 보통파일의 의미로 사용된다.

파일소유권(file ownership)
파일속성중의 하나. 파일을 만들거나 복사하고 있는 사용자가 보통 그 소유자이다. 파일의 소유자는 다른 사용자들을 거절하는 일정한 권한을 가진다. 소유권은 다른 사용자들에게 넘겨 질수 있으나 일단 넘겨 주면 회복될수 없다.

파일속성(file attribute)
파일의 특성을 서술하는 색인마디안에 저장되는 파라메터들의 묶음. 형태와 소유권, 허가권, 시간정보, 크기, 련결의 수, 15개의 디스크블로크 주소들의 배렬로 구성된다.

파일손잡이(filehandle)
perl에서 프로그람명령문들에 의하여 사용되는 파일의 물리적이름. 절환된 입출력을 이름 짓는데 사용될수 있다. write, print, printf문들은 파일손잡이를 사용하여 물리적인 파일이나 관흐름에 출력을 써넣는다.

파일시간도장(file time stamps)
파일의 최종변경 및 접근, 색인마디의 변경시간을 표현하는 3개의 날자 및 시간의 묶음. 이 정보는 색인마디안에 저장되며 ls지령으로 볼수 있다.

파일전송규약(File Transfer Protocol: FTP)
두개의 원격기계사이에 파일들을 전송하는 TCP/IP 응용. 그를 실현하는 지령의 이름도 ftp이다.

파일체계(file system)
자기의 별도의 뿌리등록부를 가지는 파일들과 등록부들의 계층적인 구조. 모든 하드디스크는 적어도 한개의 파일체계를 가지며 mount지령으로 주파일체계에 태운다.

파일허가권(file permission)
파일의 읽기, 쓰기,실행허가를 결정하는 3개씩 묶은 보호체계. 이 허가권의 묶음은 사용자의 3가지 종류 즉 사용자(user), 그룹(group), 다른 사용자(others)의 매 종류에 관해서도 유효하다. 파일의 소유자만이 chmod지령을 가지고 허가권을 바꿀수 있다.

포구번호(port number)
TCP/IP봉사를 식별하기 위하여 사용되는 번호이며 /etc/services에 정의된다. 파케트는 통로의 량끝을 위한 2개의 포구번호를 가진다. 의뢰기포구번호는 우연적인 방법으로 할당된다. ⇨ 소케트

표준출력(standard output)
문자흐름으로서 출력을 내보내기 위하여 지령들이 사용하는 목적지. 말단으로 출력을 내보내는 모든 UNIX지령들이 사용한다. 기본목적지가 또 다른 파일이나 관흐름으로 출력을 전환하도록 방향절환될수 있다.

표준오유(standard error)
진단(오유)출력흐름이 자기의 출력을 써넣기 위하여 사용하는 목적지. UNIX지령들에 의하여 발생되는 모든 오유통보문들을 포함한다. 이 흐름의 기본적인 목적지는 말단이지만 임의의 파일에로 방향절환될수 있다.

표준입력(standard input)
지령에로의 입력을 목적으로 문자흐름으로서 정보를 받아 들이기 위하여 쉘이 열어 놓은 원천. 기정적으로는 건반이 할당되지만 파일이나 관흐름으로부터 들어 올수도 있다.

프로필(profile)
매 사용자에 의하여 리용되고 관리되는 시작파일. 홈등록부안에서 .profile, .login, .bash_profile의 이름을 가진다. 이 파일에 포함된 모든 명령문들은 보조쉘을 실행함이 없이 가입시에 실행된다.

프로쎄스(process)
실행되고 있는 프로그람의 실체. 보통 지령을 실행시키기 위하여 요구된다. 쉘내부지령들의 대부분은 프로쎄스를 생성함이 없이 실행된다.

프롬프트(prompt)
유표의 위치를 보여 주는 하나이상의 문자들로 구성된 문자렬. 일반적으로 프롬프트의 출현은 종전지령이 실행을 완료하였다는것을 보여 준다. 프롬프트는 PS1 또는 prompt변수의 값을 설정하는것으로써 바꿀수 있다.

페지화도구(pager)
한번에 한개 화면분의 출력을 현시하는 도구. Linux와 UNIX체계들에서 more와 less가 표준적인 페지화도구들이다. 둘 다 탐색기능을 제공하며 vi편집기를 리용하여 파일을 편집할수 있게 한다. 원래의 페지화도구인 pg는 이제는 사멸되였다.

하이퍼본문(hypertext)
꼬리표 <A HREF>로 문서안에 배치된 련결. 이 꼬리표는 같은 기계나 다른 기계안에 있는 또 다른 문서의 어떤 위치를 지정한다. World Wide Web(WWW)는 이러한 문서들의 집합이다. ⇨ Web페지

하이퍼본문전송규약(HyperText Transfer Protocol: HTTP)
WWW상의 주콤퓨터들로부터 HTML문서들을

얻기 위하여 TCP/IP통신규약탄창의 맨 꼭대기에 위치하는 통신규약. 한 접속이 그이전 접속의 상태에 대한 정보를 가지지 않는 무상태(stateless)규약이다. HTTP 1.1은 영구접속(persistent connection)을 지원한다.

하이퍼본문표식달기언어 (HyperText Markup Language:HTML)
Web문서를 만들기 위한 다목적언어. 다른 기계에 있는 다른 문서에로 조종을 련결할수 있는 꼬리표의 존재로 하여 그 특성이 나타난다. HTML문서들은 동화상을 보거나 음악을 연주하는데 사용될수 있다.

현재등록부(current directory)
cd지령을 인수와 함께 사용한후에 사용자가 있게 되는 등록부. 가입시에는 보통 홈등록부로 설정된다. ⇨ 홈등록부

홈등록부(home directory)
사용자가 가입시에 배치되는 등록부를 가리키기 위하여 /etc/passwd안에 지정된 마당. cd지령을 인수없이 사용할 때도 리용된다. ⇨ 가입등록부

홈페지(home page)
Web싸이트에 접속할 때 사용자에게 보여 주는 첫 페지를 가리키는 용어.

흘림띠(scrollbar)
창문의 측면에 막대기모양으로 존재하는 X Window체계의 구성요소. 이 막대기우에서 마우스를 사용하여 본문을 아래우로 흘려 보낸다.

핵심부(kernel)
파일 및 프로쎄스들의 생성과 관리를 담당한 UNIX 조작체계의 부분. 기계의 하드웨어와 직접 호상작용하며 기계가 기동할 때 주기억에 읽혀 지는 파일들인 unix, genunix, vmlinuz로 존재한다.

행(line)
행바꾸기문자(newline)로 끝나는 문자들의 렬.

행내편집(in-line editing)
Korn쉘과 bash에서 유용한 기능으로서 vi형 및 emacs형지령으로 이전 지령을 재호출하여 편집한다.

행바꾸기(newline)
[Enter]건이나 [Ctrl-j]를 누를 때 발생되는 문자. 두행사이의 구분문자로 쓰이며 공백(whitespace)문자들의 하나를 형성한다.

행주소(line address)
sed, awk, perl에서 사용되는 주소화의 한가지 형식으로서 한행 또는 련속된 행들의 한 묶음을 지정한다. 하나의 행번호 또는 범위를 지정하는 한쌍의 행번호를 요구한다.

확장문자렬(escape sequence)
역사선(\)이 앞에 붙은 문자. 여기서 역사선은 그 문자에 특수한 의미를 부여한다. 확장문자렬은 echo, awk, perl에서 사용된다. 실례로 \t는 타브(tab)를 표현하며 \n은 행바꾸기(newline)를 표현한다.

확장형정규식(extended reqular expression)
다중패턴의 지정을 가능하게 하며 묶음의 사용을 허락하는 grep, awk, perl에 의하여 사용되는 확장된 정규식. 메타문자 ?, +, (,), |를 사용한다.

환경변수(environment variable)
사용자의 환경의 속성을 결정하는 쉘변수. 그것들의 일부는 쉘에 의하여 자동적으로 설정되며 사용자에 의하여 재할당될수도 있다. 그러한 변수의 값은 모든 보조쉘들에서 유효하다.

꼬리표 붙은 정규식(tagged regular expression: TRE)
\(와 \)을 리용하여 정규식을 묶음별로 분류하는것을 가리키는 용어. 이 묶음은 꼬리표 \n을 사용하여 행안의 다른 곳에서 반복된다. 여기서 n은 1과 9사이의 수이다. grep, sed, perl지령에 의하여 사용된다. perl은 또한 다음의 묶음화가 수행될 때까지 패턴들을 기억하기 위하여 \와는 별도로 $를 사용한다.

뿌리(root)
어미등록부를 가지지 않는 모든 파일체계의 최상위등록부. /기호로 표시한다. 또한 상급사용자등록자리로 가입하기 위하여 가입이름 root를 사용하는 사용자를 의미하기도 한다. /는 뿌리사용자의 홈등록부이기도 하다(Linux는 제외).

뿌리창문(root window)
X Window체계에서 전체 화면을 차지하는 기본창문. 다른 모든 창문들은 이 창문우에 현시되며 그의 새끼창문으로 간주될수 있다. 흔히 탁상화면(desktop)이라고도 한다.

뿌리이름봉사기(root name server)
BIND를 실행시키고 있는 봉사기로서 .com,.edu,.org 등 웃준위령역의 이름봉사기들을 지정한다. 그 13개의 봉사기들이 인터네트상의 모든 웃준위이름봉사기질문을 처리한다.

쪼각화(fragmentation)
TCP/IP통신규약탄창의 어떤 층에서 파케트들이 더 작은 단위로 쪼개지는 처리. 망이 작은 파케트만을 조작할수 있다는 리유로 하여 교갑화전

에 보통 쪼각화가 진행된다.

아이콘(icon)
비능동상태에서 X의뢰기를 표현하는 작은 그림. 아이콘을 두번 찰칵하는것으로써 그 의뢰기의 창문을 동작시켜 화면상에 현시한다.

암시파일(hints file)
뿌리이름봉사기의 IP주소와 FQDN을 포함하는 BIND가 사용하는 파일. BIND가 실행되고 있는 모든 기계들은 이 자료를 기억기안에 가지고 있어야 한다.

압축된 파일(zipped file)
compress, gzip, zip지령에 의하여 압축된 파일. UNIX에서 압축된 파일은 일반적으로 .Z, .gz 또는 .zip확장자를 가진다.

어미프로쎄스(parent process)
보조적인 프로쎄스들을 생성하는 프로쎄스. 보통 어미프로쎄스는 생성된 프로쎄스의 완료를 기다린다(배경에서의 실행은 제외). 어미프로쎄스를 완료하면 일반적으로 그의 모든 새끼프로쎄스도 완료된다.

열람프로그람(browser)
World Wide Web의 HTML페지들을 보기 위하여 사용하는 프로그람. 공통적인 Web열람프로그람으로서는 Netscape와 Internet Explorer가 있다. Linux는 lynx, Arena와 Netscape를 리용한다. Mosaic가 첫 Web열람프로그람이였다.

영구접속(persistent connection)
HTTP1.1에서 유용한 기능으로서 하나의 접속으로 여러 자원을 꺼내올수 있게 한다. 봉사기는 추가적인 요청을 받아 들이기 위하여 일정한 시간동안 접속을 유지한다. 영구접속은 Web접근의 속도를 높인다. 생존(keepalive)과 같다.

우편국규약(Post Office Protocol:POP)
비직결 또는 직결우편봉사기로부터 우편을 가져오기 위해 사용되는 TCP/IP규약. POP는 흔히 회선상에서 인터네트우편을 가져 오는데 리용된다. POP봉사기는 인터네트데몬(inetd)에 의하여 호출된다. Netscape는 POP의뢰기를 내장하고 있으나 Linux체계에서의 표준POP의뢰기는 fetchmail이다.

우편목록(mailing list)
가입자들의 목록을 중심적으로 관리하는 인터네트의 한가지 봉사. 주제에 따라 편성된다. 그 목록으로 주소화된 통보문은 목록의 모든 성원들에게 자동적으로 보내여 진다. listserv, listproc, majordomo들은 가장 보편적인 목록프로그람들이다.

우편배포대행체(Mail Delivery Agent: MDA)
우편을 사용자에게 배포하는것을 담당한 대행체. 우편전송대행체로부터 우편통보문을 받아 /var/mail(Linux에서는 /var/spool/mail)안에 있는 본문파일에 추가한다. mail, deliver, procmail은 공통적인 배포대행체들이다. ⇒ 우편전송대행체

우편사용자대행체(Mail User Agent: MUA)
우편을 보내고 받는데 사용되는 의뢰기프로그람. MUA는 수신된 우편의 위치를 알아 내기 위하여 /var/mail(Linux에서는 /var/spool/ mail)의 spool등록부를 검색한다. 송신하는 우편을 우편전송대행체에 넘겨 준다. mail, elm, pine과 Netscape Messenger가 공통적인 MUA들이다.

우편전송대행체(Mail Transport Agent: MTA)
망을 통하여 우편을 전송하는것을 담당한 대행체. 한 MTA가 다른 MTA에 우편을 넘겨 준후에 그 우편은 배포될수 있다. SMTP는 MTA에 사용하는 표준규약이며 그것을 실현한 가장 공통적인 실행프로그람이 sendmail이다.

우편함(mailbox)
일반적으로 /var/mail(Linux에서는 /var /spool/mail)안에 위치하는 사용자이름을 본 따서 이름 지은 본문파일로서 그 사용자에 대하여 수신된 모든 우편을 포함한다. 2진첨부물들은 부호화된 형태로 이 파일에 저장된다.

웃준위령역(top-level domain)
뿌리(.)령역아래의 모든 령역들. 이것들은 일반령역들인 com, edu, org, net 등과 두 문자의 나라별 령역(in, au, de, ca 등과 같은)을 포함한다.

이름봉사기(name server)
주콤퓨터의 FQDN을 IP주소로, IP주소를 FQDN으로 변환하기 위하여 인터네트상에서 사용되는 전용의 봉사. 분석기(resolver)가 이름봉사기에 질문을 보낸다. 이름봉사기는 그 대답을 가지고 있을수도 있고 그렇지 않으면 또 다른 이름봉사기의 주소를 제공하여야 한다.

이써네트주소(Ethernet address)
MAC주소와 같다.

인수(argument)
지령의 뒤에 놓이는 단어들. 선택항목이나 식, 지령, 프로그람, 하나이상의 파일이름일수 있다.

인터네트
① 전자우편, 파일전송, 원격가입, 망새소식,

IRC, WWW의 기능을 가진 TCP/IP통신규약에 의하여 접속된 망들의 초대형의 망(이 경우에는 Internet라고 쓴다). 세계의 모든 주요망들은 이 방식으로 접속되여 있다. ⇨ World Wide Web ② TCP/IP통신규약에 의하여 접속된 2개이상의 망들(이 경우에는 internet라고 쓴다). ftp, telnet, Web열람, 전자우편의 모든 기능들이 여기서도 유효하다. 인트라네트와 같다.

인터네트데몬(Internet Daemon: inetd)
여러 포구들을 감시하면서 다른 데몬들인 ftp, telnet와 POP봉사를 불러 내는 데몬. inetd는 봉사를 허락 또는 불허하는데 사용될수 있는 /etc/inetd.conf를 리용한다.

인터네트주소(internet address)
IP주소와 같다.

인터네트중계담화(Internet Relay Chat: IRC)
여러 사용자들이 직결식의 본문기반대화에 참가할수 있게 하는 인터네트의 한가지 봉사. 담화실(chat room)은 통로별로 나누어 지며 통로의 매 성원들이 보내는 통보문은 통로의 모든 성원들에게 실시간적으로 전달된다. irc지령은 가장 일반적으로 사용되는 IRC의뢰기이다.

인터네트통신규약(Internet Protocol: IP)
TCP/IP통신규약묶음의 중요한 구성요소로서 파케트의 경로를 조종한다. IP는 주콤퓨터에로의 경로가 아니라 망에로의 경로를 제공한다. 오유검출 및 오유정정기능은 없다.

인트라네트(intranet)
⇨ 인터네트

인용부호화(quoting)
문자들의 묶음을 그것들이 가지고 있는 특수한 의미를 제거하기 위하여 인용부호로 닫아 주는 방법. 쉘은 외인용부호(')로 둘러 막힌 모든 특수문자들을 무시하지만 겹인용부호(")는 $를 변수로 평가하는것과 `를 지령치환에 리용하는것을 허락한다.

일감조종(job control)
일감을 전경과 배경사이에서 이동시키거나 중지 또는 제거하기 위한 기능으로서 거의 모든 쉘(Bourne은 제외)에서 제공된다. 림시중지된 일감은 fg나 bg에 의하여 전경 또는 배경으로 각각 이동시킬수 있다.

입력방식(input mode)
vi편집기의 3가지 방식의 하나. 이 방식에서는 임의의 건누름이 입력으로서 해석되며 화면상에 현시된다. 이 방식은 [Esc]건을 눌러 완료한다.
⇨ 지령방식, 최종행방식

위치파라메터(positional parameters)
쉘스크립트의 외부인수. $1, $2, $3 등의 형식을 가진 일련의 특정한 변수들에 읽혀 진다. $*는 전체 문자렬을 표현하며 $#는 파라메터묶음의 개수를 의미한다.

의뢰기-봉사기구성방식(client-server architecture)
함께 작업하고 있는 두대이상의 콤퓨터들의 망구성. 의뢰기콤퓨터는 봉사기프로그람이 실행되고 있는 봉사기콤퓨터에 봉사를 요구하는 프로그람을 실행시킨다. TCP/IP와 인터네트에서 쓰이는 용어이다. X Window는 반대되는 방식으로 그 개념을 취급한다. ⇨ X봉사기, X의뢰기

의미해제(escaping)
문자의 바로 앞에 역사선(\ )을 붙여 뒤따르는 문자가 문자그대로의 의미로 취급되게 하는것. 쉘과 일부 려파기들이 이 속성을 사용한다. 대부분의 경우에 문자의 특정한 의미를 제거할뿐아니라 때때로 그것을 강조하기 위해 사용되기도 한다.

완료값(exit status)
지령이나 쉘스크립트 또는 쉘함수들이 실행후에 돌려 주는 값. 값 0은 성공적인(참) 실행을 가리키며 반면에 임의의 다른 값들은 비성공적인(거짓) 실행을 가리킨다. ⇨ 돌림값

완성(completion)
emacs편집기, bash, Korn쉘의 한가지 기능으로서 본문이 식별할수 있을 정도로 입력되면 그 전개가 자동적으로 수행되도록 한다. 쉘에서는 완성하기가 PATH에서 유용한 지령과 파일에 다 작용한다.

완전지정령역이름(fully qualified domain name: FQDN)
주콤퓨터가 속한 령역과 부분령역들을 표현하는 점으로 구분된 문자렬들의 집합. 주콤퓨터의 FQDN은 인터네트상에서 유일하다.

완충기(buffer)
자료를 저장하기 위하여 사용되는 기억기나 디스크상의 림시저장구역. vi와 emacs에서는 편집전에 파일의 복사본을 만드는데 리용된다. 완충기는 디스크의 자료를 읽거나 쓰는데, 상위블로크와 색인마디자료를 저장하는데 리용된다.

원격가입(remote login)
사용자이름과 통과암호를 리용하여 원격기계에 접속하는것. 가입후에 입력되는 모든 지령들은 실제상 원격기계상에서 실행된다. ⇨ telnet

*　　*

Apache
인터네트상에서 사용되는 가장 보편적인 Web봉사기이며 Linux체계의 표준Web봉사기. 영구적인 접속, 가상주콤퓨터기능, 등록부의 접근조종을 지원한다. httpd데몬에 의하여 표현된다. ⇨ httpd봉사기

Archie
인터네트상에서 임의의 내리적재가능한 파일들의 위치를 알아 내는 TCP/IP응용프로그람. 인터네트상에서 거의 모든 닉명ftp봉사기들을 검색하며 발견된 파일의 절대경로이름과 FQDN의 목록을 만든다. Web에 의하여 사멸되고 있다.

ASCII순서맞추기렬(ASCII collating sequence)
문자들에 번호를 붙이기 위하여 ASCII(American Standard Code for Information Interchange)가 사용하는 순서. 조종문자들이 웃부분을 차지하고 그다음은 수자, 영어대문자, 영어소문자가 놓인다. sort와 ls, wild-card들과 정규식들이 사용하는 문자모임, 출력을 정돈하는 임의의 UNIX지령이 이 렬에 의거한다.

BIND
가장 광범하게 리용되는 DNS의 한가지(현재는 BIND 8). 이름봉사를 제공하기 위하여 대부분의 UNIX체계들이 사용한다. ⇨ 령역이름체계, 이름봉사기

BSD UNIX(Berkeley Software Distribution)
캘리포니아종합대학의 버클리에서 만든 UNIX의 변종. 버클리는 후에 전반체계를 다시 서술하였으며 vi편집기, C쉘, r-편의프로그람, PPP, 기호련결과 같은 몇가지 확장을 소개하였다. TCP/IP는 BSD UNIX에서 처음으로 유효하게 되였다.

cron
UNIX체계의 크로노그라프(시간을 도형적으로 기록하는 장치). 조종파일(crontab)안에 렬거된 지령들을 파일의 여러 마당에 지정된 주기에 따라 실행시킨다. 체계관리자라고 해도 그것을 직접 불러 낼수는 없다. ⇨ crontab

crontab
주기적으로 실행될 필요가 있는 모든 지령들과 그것들의 실행주기를 포함하고 있는 사용자ID를 따서 이름 지은 조종파일. cron지령은 실행시간표가 작성된 지령을 실행시키기 위하여 매분마다 이 표를 조사한다.

DARPA
최종적으로 인터네트를 형성시킨 초기작업을 담당하였던 미국방성의 한 연구조직. DARPA에 의하여 개발된 도구들에는 telnet와 ftp도 속한다.

exec
호출된 프로쎄스의 코드로 자체를 덧쓰기 위하여 프로쎄스가 사용하는 체계호출 또는 프로쎄스의 표준입출력을 재할당하는 쉘내부명령문의 이름. 그것은 물리적인 파일이름보다 오히려 파일이름서술자를 사용하며 I/O를 조작하는데 유용하다.

export
어미프로쎄스의 환경이 새끼프로쎄스에서 유효하도록 하는 내장쉘지령. 이 명령문은 변수들과 함께 사용된다.

ex방식(- mode)
최종행방식과 같다.

FAQ
모든 화제에 관한 망새소식의 제공자들에 의하여 관리되는 자주 제기되는 질문(frequently asked question)들의 묶음.

fork
프로쎄스가 자기의 정확한 복사를 만들어 내기 위하여 사용하는 체계호출. 복사된 프로쎄스는 어미측의 열려 진 파일들, 현재등록부, 외부변수들을 물려 받지만 서로 다른 PID를 가진다.

GET
Web봉사기에로 양식자료를 보내는 메쏘드. 자료는 <이름=값>의 쌍으로 보내여 지며 Web봉사기의 QUERY_STRING변수안에서 유효하다. 이 방식을 사용할 때 질문문자렬의 크기에서 제한이 있다. ⇨ POST

getty
다음번 가입을 감시하기 위하여 모든 빈 말단들에서 실행되는 프로쎄스. 사용자가 가입을 시도할 때마다 login프로그람을 실행시킨다.

GNU
리챠드 스톨맨이 창설한 기구(그 이름은 GNU's Not UNIX로 확장된다). 현재 자유쏘프트웨어재단(Free Software Foundation)으로 알려 져 있다. Linux에서 리용하는 많은 도구들이 GNU에 의하여 개발되였으며 그의 허가밑에 배포되였다. GNU허가는 모든 개발자들이 원천코드를 공개할것을 요구한다.

here문서(here document)
<<기호를 리용하여 많은 지령들이 사용하는 표

준입력에 관한 양식. 지령에 대한 입력은 지령행 자체의 부분을 형성한다. 특히 인수로서 입력파일이름을 받아 들이지 않는 지령과 함께 사용할 때 쓸모가 있다.

httpd봉사기

Web봉사를 관리하기 위하여 httpd데몬을 실행시키고 있는 봉사기. WWW는 httpd봉사기우에서 실행된다. 봉사기는 의뢰기열람프로그람이 요구하는 모든 문서들과 그림들을 전송하며 CGI응용프로그람에로 양식자료를 넘겨 준다. 기본 httpd데몬은 매 요구를 조작하기 위한 별도의 새끼httpd데몬을 생성한다. Web봉사기와 같다. ⇨ Apache

init

PID번호 1을 가지는 프로쎄스로서 체계의 모든 주요프로쎄스들의 생성을 담당한다. init는 모든 체계데몬을 실행시키며 말단포구들에 getty프로쎄스들을 실행시킨다. /etc/inittab로부터 명령을 취한다. 그것은 또한 체계에 특정한 실행준위를 설정하기 위한 지령으로서도 사용된다.

IP주소(IP address)

TCP/IP망에서 기계의 론리적주소를 서술하기 위하여 사용되는 점으로 구분된 4개의 8비트들의 렬. 자료가 주콤퓨터에 당도하기전에 모든 FQDN들은 이 주소로 변환된다. 최종적으로는 수신측에서 이써네트주소(Ethernet address)로 변환된다. ⇨ 인터네트주소

MAC주소

망대면부기판의 48bit주소로서 세계적으로 유일하다. 모든 IP주소가 이 주소로 변환된 다음에만 수신기계가 그것을 리해할수 있다. 이써네트주소와 같다.

MULTICS

UNIX조작체계에 밀리워 그 개발작업이 류산된 조작체계. UNIX의 많은 기능들이 MULTICS에 기원을 두고 있다.

Netscape

네트스케이프회사로부터 개발된 프로그람의 묶음. 이 묶음은 Web를 열람하기 위한 Netscape Navigator와 우편 및 새소식그룹을 처리하기 위한 Netscape Messenger를 포함하는 Netscape Communicator로 알려져 있다. netscape지령으로 호출한다.

PATH

사용자가 호출한 지령의 위치를 알아 내기 위하여 쉘이 탐색하여야 할 등록부들의 목록(두점으로 구분)을 포함하는 쉘변수. PATH는 보통 권한 없는 사용자들에 관해서는 /bin과 /usr/bin을 포함하며 체계관리자에 관해서는 /sbin과 /usr/sbin을 포함한다.

ping

망의 접속상태를 검사하기 위하여 원격주콤퓨터에 파케트를 보내는것 또는 그것을 수행하는 지령의 이름.

POSIX

UNIX조작체계상에 기초한 표준대면부들의 묶음. POSIX적응은 한 기계상에서 개발된 프로그람들의 묶음이 추가적인 작업이 없이 다른 기계에 이식될수 있게 한다. POSIX.1은 C언어에 관하여 대면부프로그람을 작성하는 응용프로그람의 표준을 표현하며 POSIX.2는 쉘과 도구프로그람들에 관한 대면부를 제공한다.

POST

Web봉사기에로 양식자료를 보내는 한가지 메쏘드. 자료는 <이름=값>형태의 문자렬로서 전송되며 표준입력으로서 CGI프로그람에 공급된다. 이 방법을 사용할 때는 문자렬의 크기에 관하여 제한이 없다. ⇨ GET

rc스크립트(rc script)

/etc안의 rcn.d등록부(여기서 n은 실행준위를 가리킨다.)안에서 유용한 쉘스크립트들의 묶음으로서 기계에 특정한 실행준위를 설정하는데 사용된다. 이 스크립트들은 그 준위에서 실행되여야 할 데몬들을 실행시키며 실행되지 말아야 할 데몬들을 제거한다. ⇨ 시작스크립트, 정지스크립트

r-편의프로그람(r-utilities)

DARPA묶음을 대신하기 위하여 버클리에서 개발한 TCP/IP도구의 묶음. rlogin, rcp, rsh가 포함된다. 이 도구들을 통한 원격체계에로의 접근은 원격기계상에 있는 /etc/hosts.equiv와 $HOME/.rhosts에 의하여 조종된다.

sendmail

SMTP통신규약을 실현한 가장 공통적인 프로그람. sendmail은 우편을 받기 위해서는 데몬방식으로 실행되며 우편을 보내기 위해서는 의뢰기로서 실행된다. 구성은 /etc/sendmail.cf에 의하여 조종된다. sendmail은 또한 임의의 주콤퓨터상에 있는 임의의 사용자에게 우편을 발송하기 위하여 /etc/aliases를 사용한다.

TCP/IP

전송조종규약(Transmission Control Protocol)/

인터네트통신규약(Internet Protocol)의 략자로 서 여러가지 조작체계와 여러가지 하드웨어를 사용하는 콤퓨터들을 망으로 구성하기 위하여 사용되는 통신규약들의 집합이다. 완전한 오유정정기능으로 하여 믿음성 있는 전송을 담보한다. 여기서 TCP와 IP는 가장 중요한 통신규약이다.

telnet

사용자가 사용자이름과 통과암호를 제공한후에 원격기계에 가입할수 있게 하는 TCP/IP응용프로그람. 가입후 사용자는 그 원격기계를 마치 국부기계인것처럼 사용할수 있다. 모든 파일들이 원격기계상에 만들어 진다.

The Open Group

UNIX표준의 소유자와 UNIX98명세의 설계자. X/OPEN을 포함한다. X Window체계도 관리한다.

URL(Uniform Resource Locator)

Web자원에 접근하기 위하여 Web열람프로그람의 별도의 창문(주소창문)에 입력되는 문자렬. 통신규약, 싸이트의 FQDN, 파일의 경로이름으로 구성된다. URL에서 사용되는 표준적인 통신규약접두사는 http://이지만 ftp://나 telnet://도 될수 있다.

Web봉사기(Web server)

httpd봉사기와 같다.

Web페지(Web page)

모든 Web싸이트에서 페지의 형식으로 존재하며 열람프로그람으로 보여 지는 HTML문서. 본문, 그림, 동화상을 포함하며 음성과 비데오자원을 지정할수 있다. 한 페지는 다른 페지들에 대한 련결을 가진다(때때로 다른 기계상의 페지에로). 페지들은 사용자입력을 위한 양식을 생성하는데 리용될수 있다.

Web싸이트(Web site)

httpd봉사를 주관하는 하나의 기관에 속한 하나 또는 그이상의 기계. 그 기관과 관련된 모든 문서들과 기타 파일들을 관리한다. 관련정보에 접근하기 위하여 다른 Web싸이트에 대한 련결도 관리할수 있다.

World Wide Web

련결된 문서들과 화상들의 집합을 특징 짓는 인터네트상의 봉사. Web는 HTTP통신규약을 사용하며 페지들을 현시하기 위하여 열람프로그람을 요구한다. 하이퍼본문은 사용자들이 마우스조작만으로 다른 문서에로 넘어 갈수 있게 해준다. 현재 인터네트의 가장 활력 있고 유효한 분야이다.

X Window

UNIX체계의 도형요소. X의뢰기들은 봉사기에 자기들의 출력을 써보내며 봉사기는 분리된 창문들에 그것을 현시하는것을 책임진다. Netscape는 우편을 조작하고 Web를 열람하기 위하여 이 체계를 요구한다.

X봉사기

X Window에서 감시기, 건반, 마우스를 포함한 현시장치를 조종하는 프로그람. X의뢰기들은 자기들의 출력을 이 프로그람에로 써보낸다. 현시장치가 변화되면 그 봉사기만이 변화될 필요가 있을뿐 의뢰기들은 그럴 필요가 없다.

X의뢰기

특정한 기능을 수행하며 현시를 위하여 X봉사기를 사용하는 X프로그람. xterm과 xclock가 모든 X Window체계에서 볼수 있는 일반적인 X 의뢰기들이다.

* *

64진수

2진첨부물을 본문형식으로 변환하기 위하여 현대우편처리기(mailer)들이 사용하는 부호화형식. 3byte의 자료를 4개의 6bit문자로 변환하며 파일의 용량을 3분의 1만큼 증가시킨다.

부록 8. 시험문제의 답

1장

1. 응용프로그람과 사용자
2. ASCII값
3. [Ctrl] 및 [Alt]건
4. 기계의 이름
5. 할수 있다. 그러나 그런 방법은 피하여야 한다.
6. 필요 없다. 통과암호가 부정확할수도 있다.
7. [Ctrl-c]를 사용하며 동작하지 않을 때에는 [Del]이나 [Delete]를 사용한다.
8. who am i 또는 whoami를 사용한다.
9. ls -l을 사용한다.
10. 파일의 내용을 현시하는데는 cat가 사용된다. 두개의 파일이름이 사용될 때에는 첫 파일의 내용을 현시한 다음 둘째 파일을 현시한다.
11. 켄 톰슨과 데니스 리치에
12. 왜냐하면 그때 콤퓨터쏘프트웨어를 판매하는 것이 정부에 의하여 금지되였기때문이다.
13. 캘리포니아종합대학의 버클리에서.
14. Solaris와 SunOS
15. SCO UNIX와 Intel Solaris, Linux.
16. X/Open, 현재의 Open Group의 부분.
17. 리챠드 스톨맨과 리누스 토발즈
18. 쏘프트웨어개발자들이 원천코드를 공개해야 한다는 허가밑에서 제품을 배포하것.
19. 핵심부
20. 고급언어인 C로 개발되였기때문이다. 고급언어로 씌여 진 프로그람은 기본적인 변경이 없이 다른 기계상에서 실행될수 있다.
21. System V(AT&T) 와 BSD(Berkeley). SunOS는 BSD에 기초하고 있지만 Solaris는 AT&T의 SVR4에 기초하고 있다.
22. (1) unix 또는 genunix (2) vmlinuz
23. 다중과제처리는 사용자가 한번에 한개이상의 일감을 실행시킬수 있다는것을 의미한다.
24. 그러한 간단한 일감들을 여러개 련결하여 복잡한 일감들을 조작할수 있기때문이다.
25. X Window.
26. 쉘과 perl, tcl, python.
27. Red Hat, SuSE, Caldera.

2장

1. 두점(:)이 아무것도 하지 않는 지령 즉 빈 지령이여야 한다.
2. 일반적으로는 아니다. 즉 UNIX는 대소문자를 구분한다.
3. 물론이다. UNIX는 확장자에 대하여 주의를 돌리지 않는다.
4. mkdir와 rmdir.
5. PATH는 쉘이 지령의 위치를 검색하기 위한 등록부들의 목록을 저장하는 체계변수이다.
6. 두점(:)
7. 아니다. 지령들과 인수들은 공백에 의하여 구분되여야 한다. 그렇지만 ls -l의 별명으로서 ls-l을 참조할수 있다.
8. 3개의 선택항목(-l, -u, -t).
9. Linux지령들이 흔히 선택항목앞에 두개의 - 기호를 사용한다.
10. 때때로 그것들은 +를 앞에 붙이거나 =를 포함할수 있다.
11. 때때로 일부 지령들은 선택항목이 없이 실행될수는 없다(cut와 같이).
12. 지령행.
13. awk와 perl.
14. uname -r를 사용한다.
15. 그 지령들이 사용자에 의하여 자주 리용되며 더빨리 위치를 알아 낼것이기때문이다.
16. 아니다. 지령들이 타자앞(type-ahead)완충기에 저장되며 자동적으로 순서에 따라 실행될수 있다.
17. man이 지령이름과 함께 사용될 때 화면에 현시되는 그 지령의 문서.
18. info지령.
19. apropos를 열쇠단어와 함께 사용하여 본다.

3장

1. 체계관리자가 지령을 불러 낼 때.
2. 파일 /etc/shadow안에.
3. 조정목적으로 9월에 11일을 뛰여 넘었다.
4. 오직 체계관리자인 경우에만.
5. calendar는 사용자의 현재의 등록부에서 파일 calendar로부터 입력을 가져 온다.

6. clear와 tput clear.
7. 조작체계의 이름.
8. w지령.
9. script지령을 인수(파일이름)와 함께 사용한다.
10. tty지령을 사용한다.
11. lock지령을 사용한다(Solaris에서는 무효).
12. scale=3으로 설정한다.
13. w henry를 사용한다.
14. users지령을 사용한다(Solaris에서는 무효).

4장

1. R를 사용하여 본문을 입력하고 [Esc]를 누른다.
2. 1G로서 첫행에로 이동한 다음 영문자 O를 누른다.
3. 유표를 d에로 옮긴 다음 rt를 사용한다.
4. S를 사용하고 새로운 본문을 입력한후 [Esc]를 누른다.
5. 최종행방식지령 :q!를 사용한다.
6. 40|를 사용한다.
7. 6k를 사용한다.
8. Ctrl-l]을 사용한다.
9. .w foo[Enter]를 사용한다.
10. 10yw를 사용한다.
11. 그것들은 연산자들이며 다른 지령들의 결합에 사용된다.
12. :1,$s/Internet/Web/g를 사용한다.
13. d1G를 사용한다.
14. :e!를 사용한다.
15. 구역(region), 다중창문, 리력기능(최종행방식에서)
16. [Ctrl-r]를 사용한다.

5장

1. 완충기가 수정되지 않는다는것.
2. [Ctrl-g]를 사용하여 지령을 취소한다.
3. 그것이 도움말기능을 호출하기때문이다.
4. [Alt-x]overwrite-mode[Enter]를 사용한다.
5. [Ctrl-x][Ctrl-w]와 그다음 foo2를 입력한다.
6. [Ctrl-k]를 사용한다.
7. [Ctrl-a][Ctrl-k]를 사용한다.
8. [Ctrl-u][Ctrl-u][Alt-d]를 사용한다.
9. [Ctrl-a][Alt-40][Ctrl-f]를 사용한다.
10. [Ctrl-v](전방)와 [Alt-v](후방)을 사용한다.
11. [Alt->]를 사용한다.
12. 먼저 [Ctrl-e]를 사용하여 행의 끝으로 이동한 다음 문자렬을 입력한다.
13. 질문치환기능 즉 [Alt-%]를 사용한후 문자렬을 입력한다.
14. [Alt-x]revert-buffer[Enter]를 사용한다.
15. [Alt-x]replace-string[Enter] 를 사용하고 그다음 두개의 문자렬을 입력한다. replace-string을 입력할 때에는 지령완성하기기능 repl[Tab]s[Tab]를 사용한다.
16. [Alt-!]를 사용한 다음 date지령을 사용한다.

6장

1. 콤퓨터의 주기억기.
2. 현대적인 체계들에서는 255개의 문자.
3. 예. UNIX는 대소문자를 구별한다.
4. 뿌리등록부.
5. 체계는 항상 모든 등록부안에 이 파일들을 가지고 있으며 그것을 만드는것을 거절한다.
6. 행바꾸기(newline)문자.
7. 문법에 꼭 맞으며 마지막 bar는 보통파일이여야 하며 나머지는 등록부여야 한다.
8. 그것은 간단히 홈등록부로 절환된다.
9. 예. 현재의 등록부아래에 만들어 진 사슬로서 등록부 cd, mkdir, rmdir를 가질수 있게 한다.
10. 예. 어미등록부안의 모든 파일들을 보여 준다.
11. 그들중의 일부만. 다른것들은 간단히 작업하지 않는다.
12. man mkdir는 mkdir -p share /man/cat1의 사용을 암시할것이다. -p선택항목은 모든 보조등록부들을 한번의 실행으로 만들것이다.
13. 아니다. 그것을 삭제하기전에 그우로 이동하여야 한다. 그렇지만 Linux는 그것을 허용한다. 즉 그 등록부자체에서 rmdir.을 사용할수 있다.
14. cp -r bar1 bar2를 사용한다.
15. 확실하게 그렇게 될것이다. mv는 등록부들도 이름을 바꿀수 있다. 그러나 bar2가 존재한다면 bar1은 bar2의 보조등록부로 된다.
16. cat -v foo를 사용한다.
17. 파일들은 블로크형과 문자형으로 나누어 질수 있다.
18. gzip로 압축된 파일.

7장

1. 먼저 수자, 그다음 대문자, 소문자.
2. -F선택항목은 등록부들을 /로 표시하고 실행프로그람들을 *로 표식한다.
3. 점으로 시작하는 이름.
4. ls -l지령의 출력.
5. ls -R/를 사용한다.

6. 어미등록부의 파일들을 렬거하기 위하여 ls ..을 사용한다.
7. 허가권마당의 첫 문자로서 d.
8. 오직 파일의 소유자나 체계관리자만.
9. 기정으로 숨은 파일들은 보여 주지 않는다. 그것들을 현시하려면 -a선택항목도 사용해야 한다.
10. 파일이 1년이상 오래된것일 때(Linux에서는 6개월).
11. 그 파일이름을 포함하는 등록부상에 그룹쓰기허가를 주어야 한다
12. 다음의 두가지 방법으로 가능하다.
(1) chmod 700 foo
(2) chmod u+x,go-r foo
13. 등록부는 그들에 대한 쓰기허가권을 가진다.
14. ls -i를 사용하라.
15. /etc/passwd와 /etc/group안에.
16. 실례로 henry에게 이전시키려고 한다면 chown -R henry *를 사용한다.
17. touch -m 09301030 foo를 사용한다.
18. 최종변경 및 접근시간으로서 현재의 체계날자와 시간을 가지고 파일 foo를 만든다. 파일이 존재한다면 이 시간정보를 변경시킨다.
19. 허가권, 련결, 소유권, 시간도장과 같은 모든 파일속성(파일이름은 제외).
20. 파일은 3개의 이름을 가지지만 디스크상에는 하나의 복사본만을 가진다.
21. rm지령으로.
22. find / -name "*.html" -o-name "*.java" -print를 사용한다.

8장

1. 그것들이 그 지령에 대하여 아무것도 의미하지 않기때문이다.
2. 현재등록부안의 모든 파일을 정합하도록 그것을 전개한다.
3. chap[a-cx-z]를 사용한다.
4. 점으로 시작되는것들은 제거하지 않는다.
5. ls -???*를 사용한다.
6. find지령은 -name열쇠단어에 대한 파라메터로서 통용기호표시를 사용한다.
7. 내용이 덧씌여 진다.
8. 문제가 없다. 그 파일은 만들어 진다.
9. 적당한 위치에 그 행들을 갈라 놓지만 [Enter]건을 누르기전에 \를 입력하시오.
10. 이것은 bc < bar > foo를 실행시키는 또 하나의 방법이다. 그것은 합법적인 지령이다.
11. 2>를 가지고 표준오유를 /dev/null에로 절환한다.
12. 불가능하다. |와 more는 ls의 인수로 간주될 수 없다.
13. 두번째 경우에 변수의 평가는 외인용부호들에 의하여 방지된다.
14. 관흐름으로 두개의 지령을 사용한다. 즉 who | wc -l.
15. 아니다. 쉘은 x를 지령으로서 해석하며 = 와 10은 그의 두 인수로서 해석하다.
16. 처음것은 변수 directory를 문자렬 pwd로 설정한다. 둘째것은 그것을 현재등록부의 절대경로이름으로 설정한다.
17. bash는 Linux에서 표준쉘이다.
18. echo와 함께 -e선택항목을 사용하지 않았기 때문이다.

9장

1. vi편집기를 위하여 v를 누른 다음에.
2. /Internet[Enter]를 사용한다.
3. n을 반복적으로 사용한다.
4. space, tab, newline을 포함하지 않는 문자들의 순서렬.
5. lines=`wc -l foo` 또는 set lines=`wc -l foo`(C쉘)를 사용한다.
6. 이 문자들로 파일을 만들고 그다음 od -bc foo를 사용한다.
7. pr -t -d foo를 사용한다.
8. 두 파일이 동일하다는것.
9. comm -12 foo1 foo2를 사용한다.
10. echo "The line length of shortlist is `head -1 shortlist | wc -c`"를 사용한다.
11. head는 매 파일로부터 10개 행을 꺼내지만 파일이름을 보여 주는 머리부를 매 묶음의 앞에 붙인다.
12. tail의 많은 판본들이 반대순서로 행을 표현하기 위하여 -r선택항목을 사용한다.
13. tail -f install_log.lst를 사용한다.
14. 아니다. -c선택항목 또는 -f선택항목이 지정되여야 한다.
15. 가능하다. 그러나 실행될 지령은 paste foo1 foo2이다.
16. sort -u foo를 사용한다.
17. sort -t "|" +4.0 shortlist를 사용한다.
18. DOS파일에서는 행이 자리복귀 및 행바꾸기(CR-LF)문자에 의하여 끝난다. UNIX파일은 행바꾸기문자(LF)를 사용한다.

19. spell(System V)과 ispell(Linux)를 사용한다.

10장

1. 그것들은 fork와 exec이다.
2. echo $$를 사용한다.
3. 쉘스크립트가 실행되고 있을 때.
4. getty프로쎄스.
5. init, lpsched, inetd, cron은 말단과 협력하지 않는다.
6. ps | tail +2를 사용한다.
7. 그것이 사용자가 작업하고 있는 파일의 이름을 보여 주기때문이다.
8. 오직 체계관리자일 때에만.
9. -9 또는 -SIGKILL선택항목.
10. 오직 체계관리자일 때에만.
11. kill -l을 사용한다.
12. 사용자가 아마 일감조종을 지원하지 않는 Bourne쉘을 사용하였을것이다.
13. 먼저 쉘프롬프트에로 오기 위하여 [Ctrl-z]를 누른다. fg를 입력하여 vi에로 되돌아 간다.
14. 그 일감의 이름을 알수 없다.
15. at 8 pm tomorrow < dial.sh를 사용한다.
16. 새치기건을 사용한다([Ctrl-d]가 아닌).
17. 둘 다 가지고 time지령을 사용하며 실시간에 주목한다.

11장

1. 주콤퓨터이름의 현시와 설정.
2. 소케트는 4개의 수 즉 의뢰기와 봉사기의 IP주소 및 포구번호의 묶음으로 구성된다.
3. 그 파일은 망의 모든 주콤퓨터들에서 관리되여야 한다.
4. 완전지정령역이름(FQDN). 점으로 분리된 령역문자렬들로 구성된다.
5. 예. DNS는 대소문자를 구별한다.
6. 우편을 처리하기 위한 sendmail, Web봉사를 위한 httpd, ftp와 telnet을 위한 inetd.
7. talk brenda@uranus를 사용한다.
8. 파일 .plan과 .project안에 세부정보를 남긴후에 떠나야 한다.
9. finger @saturn을 사용한다.
10. [Ctrl-]]을 사용한다.
11. 기계의 이름을 현시하기 위하여 uname -n 또는 hostname지령을 사용한다.
12. 열람프로그람창문에 URL로서 다음의 문자렬을 입력한다.
    telnet://saturn.planets.com
13. 두 기계상에 동일한 사용자등록자리를 가지는것.
14. 아니다. ftp는 /bin/ls를 사용한다.
15. lcd bar를 사용한다.
16. 닉명ftp싸이트에는 파일을 올리적재할수 없다.
17. 원격주콤퓨터 jupiter상에서 date지령을 실행시키고 그 출력을 파일 .date안에 원격으로 저장한다.

12장

1. X는 모든 화소가 개별적으로 조작될수 있는 비트배렬표시장치를 사용하기때문이다.
2. startx와 xinit이다. 뿌리사용자일 때에만 xdm을 사용할수 있다.
3. 뿌리창문우에 마우스지시자를 옮기고 마우스의 어느 한 단추를 누른다
4. X봉사기.
5. 예. X봉사기가 HP-UX상에서 실행되고 있는 조건에서
6. xhost는 사용자가 다른 기계상에 쓸수 있게 또는 쓸수 없게 한다. xhost +는 모든 의뢰기들에 대한 접근을 허용한다.
7. 제목띠(title bar)의 색갈에 의하여. 오직 하나의 창문만이 이 색갈을 가질것이다.
8. [Alt]를 누른 다음 창문의 제목띠를 끌어 옮기는것으로서 그 창문을 이동시킨다. 창문은 선택되지 않지만 그래도 움직인다.
9. 검사칸은 여러개의 선택선택항목을 허락하지만 단일선택단추는 그렇지 못하다.
10. 임의의 창문의 제목띠에서 마우스의 왼쪽단추를 누른다.
11. 그것은 CDE가 출현하기전에 UNIX체계들의 표준으로 되였던 Motif창문관리기이다.
12. xterm -sb -sl 1000을 사용한다.
13. $HOME/.xinitrc안에 xterm &을 배치한다.
14. 그 사용자가 호출한 의뢰기를 위한 자원설정.

13장

1. 우편이 추가되는 보통파일.
2. 통보문이 현시되였다는것을 의미한다.
3. 등록부 /var/mail(SVR4) 또는 /var/spool/mail(Linux)로부터.
4. 처음것은 사용자에게 우편을 보내며 둘째것은 우편함을 현시한다.
5. 실례로 charlie에게 보여 주려고 한다면 ps -e

| mail charlie를 사용한다.
6. $HOME/.elm/elmrc안에.
7. 자기의 통보문을 구성하고 [Ctrl-j]를 사용한 다음 파일이름을 입력한다. 유표가 머리부구역에 있어야 한다.
8. 그것들이 다매체파일들을 첨부물로서 조작할 수 있기때문이다.
9. .forward파일을 제거한다.
10. 우편배포대행체(MDA)에.
11. xterm창으로부터 netscape -messenger &를 사용한다.

14장

1. com, net, org들이다.
2. 보낼수 없다. 이것은 관리주소이다. 통보문들은 그 목록의 주소에로 지정되여야 한다.
3. 새소식통보문들은 매우 빨리 구성되므로 관리자는 낡은것들을 지워 버림으로써 새 통보문들을 위한 공간을 만들어야 한다.
4. misc.test에로 통보문을 보낸다.
5. 새소식그룹의 자주 제기되는 질문(FAQ)들을 포함하는 문서.
6. 두 사용자가 같은 IRC망에 있을 때에만.
7. 그것이 도형을 처리하지 않기때문이다. 사실 도형들은 넓은 망대역너비를 소비한다
8. 포구번호 80에서 HTTP규약.
9. 마우스왼쪽단추로 Back단추를 누르고 있는다. 차림표가 이미 방문했던 모든 페지들을 보여주며 거기서 요구되는 페지를 선택해야 한다.
10. 그 싸이트에 서표를 붙인다.
11. 아니다. 하이퍼본문련결은 그림을 가리킬수 있으며 심지어 그림이 또 다른 그림을 가리킬수 있다.
12. 일반적으로 파일 index.html이다.
13. FQDN의 자리에 일반이름 localhost를 사용한다. 즉 http://localhost.
14. 한 접속이 이전 접속의 정보를 가지고 있지 않기때문이다.
15. 그것은 Content-Type: text/html이다.
16. 그 언어가 단순히 내용의 의미를 지정하는것으로써 열람기에 충고하기때문이다.
17. 도형의 서로 다른 부분들이 서로 다른 자원을 지정하는 하이퍼련결(hyperlink)로서 동작하는 도형.
18. 짐작되는 도형우에 유표를 놓고 오른쪽단추를 누른다. 선택항목 Save Image As가 나타나면 그것은 도형이다.

15장

1. grep는 파일 b와 c로부터 a를 검색한다.
2. count=`cat foo[123] | grep -c "HTML"`을 설정한다(C쉘사용자들은 이 명령문의 앞에 set를 붙여야 한다).
3. grep -li "a href" *를 사용한다.
4. 대체로 그렇다. 그렇지 않으면 그 쉘은 파일이름 TABLE을 열려고 할것이다.
5. 오직 Linux에서만. grep -A 5 "<TABLE>" foo를 사용한다.
6. 한번이상의 g의 발생.
7. grep "harriso*n*" foo를 사용한다.
8. a로 시작하여 b로 끝나는 가장 긴 패턴문자렬로써 될수록 행의 왼쪽으로부터 가장 가까운것.
9. (1) grep "................" foo(16개 점)를 사용한다.
   (2) grep ".\{16\}" foo를 사용한다.
10. egrep "(lock|har)wood" foo를 사용한다.
11. 탐색식이 정규식이 아니고 간단한 문자렬일 때.
12. 지령행에서 사용된것과 꼭 같은 방법으로 파일을 패턴들로 채운 다음 그 지령을 -f선택항목과 함께 사용한다.
13. sed 'p' foo를 사용한다.
14. sed -n '$!p' foo를 사용한다.
15. 아래의 지령을 사용한다.
   sed 'a\

   foo
16. sed 's/Linux/Red Hat &/g' foo를 사용한다.
17. grep UNIX foo | tail -1을 사용하여 파일 foo안에서 마지막으로 발생하는 패턴 UNIX를 탐색한다.

16장

1. 그것들은 같다. 둘 다 전체 행을 출력한다.
2. sort지령은 겹인용부호안에 놓여야 한다.
3. awk '{ x= NR % 2 ; if (x == 1) print }' foo를 사용한다.
4. 아래의 지령을 사용한다.
   awk -F"|" 'substr($5,0,2) == "09" || substr($5,0,2) == "12"' empn.lst.
5. awk 'length > 100 && length < 150' foo를 사용한다.
6. ls -l | awk '{tot += $5} END { print tot}'를 사용한다.
7. 지령치환을 사용한다.

x=`awk -F"|" '{ x+= $6 } END { print x/NR }' emp.lst`

8. 아래의 관흐름을 사용한다.

```
echo "DOCUMENT LIST" | \
awk '{ for (k=1 ; k < (55 ? length($0)) / 2 ; k++)
    printf "%s"," "
    print $0 }'
```

17장

1. 파일 vt220은 /usr/lib/terminfo/v안에 있다.
2. mail(C쉘)이나 MAILCHECK(다른 쉘)가 지정한 수만큼의 빈도로.
3. set PAGER = less (C쉘) 또는 PAGER=less(다른 쉘)을 설정한다.
4. 변수 user(C쉘)나 LOGNAME(다른 쉘).
5. set -o noclobber(bash나 Korn) 또는 set noclobber(C쉘)를 설정한다.
6. 아니다. 보조쉘들은 오직 환경파일만을 읽는다.
7. export지령에 의하여 모든 보조쉘들에서 유용하게 되거나 setenv문(C쉘)으로 정의될 때.
8. 오직 .bash_profile만.
9. (1) set history = 200을 사용한다.
   (2) HISTSIZE=200을 사용한다.
   (3) Korn은 오직 파일안에만 지령들을 저장한다.
10. (1) !!　(2) r　(3) !!.
11. (1) set -o vi 또는 VISUAL=vi 또는 EDITOR=vi중의 하나.
    (2) set -o vi만.
12. 실례로 henry의 경우라면 cd ~henry를 사용한다.
13. C쉘과 bash에서 ih는 경로이름의 등록부부분(head)을 떼여 내며 it는 기초파일이름(tail)을 떼여 낸다.

18장

1. 값은 x10$와 1010이다.
2. 등록부안에 같은 이름을 가진 프로그람이 있으며 그 등록부는 PATH목록안에서 현재것보다 더 먼저 놓여 있다. 현재등록부의 스크립트는 전혀 실행되지 않았다. 정확한 스크립트를 실행시키려면 ./foo.sh를 사용한다.
3. 스크립트의 첫행에 #!/bin/ksh문을 사용한다.
4. 쉘은 조작할수 있을 때까지 보조쉘들을 련속적으로 생성할것이다.
5. $#는 5이며 $*는 존재하는 모든 파일들이 제공되는 문자렬 -l -t bar1 bar2 bar3에로 확장된다.
6. 완료값은 지령의 성공이나 실패를 표현하는 옹근수이다. 지령이 성과적으로 실행되였을 때는 값 0을 가지며 파라메터 $?에 저장된다.
7. 참이다. 실행할것이 없기때문에 오유는 사용자가 무엇인가를 할 때에만 만들어 진다.
8. 패턴의 위치를 찾을수 없을 때 오직 grep만이 실패완료값을 돌려 주며 다른것들은 참의 값을 돌려 준다.
9. 외부지령들은 expr, sleep, basename들이다. 그것들은 쉘스크립트내부에서 사용될 때 가장 쓸모 있다.
10. 예. 쉘은 표준입력을 받아 들인다. 사실상 쉘은 거의 모든 시간동안 그것을 사용한다.
11. 그 문자렬과 파일이름을 포함하는 2개 행을 가진 파일 foo를 만들고 그다음 emp4.sh<
12. foo를 사용한다.
13. 스크립트안에서 bc < $1 | paste -d= $1 - 을 사용한다.
14. 0보다 큰 값이 참으로 간주된다는데로부터 이것은 무한순환을 설정한다.
15. for에 대한 목록으로서 10개의 단어를 제공한다. 지령 cmd는 10번 실행될것이다.

```
for x in 0 1 2 3 4 5 6 7 8 9 ; do
  cmd
done
```

16. 변수 x는 순환내부에서 사용되지 않는다. 보통 그 변수는 내부에서도 사용되지만 그것을 사용하기 위한 지시가 없다.

19장

1. 위치파라메터안에 set지령자체의 출력을 저장한다.
2. Korn쉘과 bash쉘에서는 ${12}를 사용한다. Bourne쉘에서는 shift 3을 사용하고 그다음 $9에 접근한다.
3. grep가 패턴 A HREF에 대한 탐색에서 실패하면 set는 인수없이 동작하며 말단상의 모든 쉘변수들을 현시할것이다.
4. 현재의 등록부는 새끼프로쎄스에 의하여 상속된 환경파라메터의 하나이다. 새끼프로쎄스가 어미프로쎄스의 환경을 바꿀수 없다는데로부터 보조쉘에서 실행된 스크립트의 제거후에 원래의 등록부가 회복되여야 한다. 영구적인 변화를 만들려면 스크립트를 .지령과 함께 실

행시킨다.

5. export로 그 변수를 내보낸다.
6. echo는 아무것도 보여 주지 않는다. script는 보조쉘에서 실행되며 거기서 정의된 변수의 값은 어미쉘에서 유효하지 않다.
7. 그 지령은 아마 Bourne이나 C쉘에서 실행되였을것이다. 그것은 bash나 Korn에서 사용될 때에는 $x안에 저장된 문자렬의 길이를 평가한다.
8. 스크립트가 인수없이 호출된다면 $1은 비여 있을것이며 flname은 emp.lst로 설정될것이다.
9. 입력의 규모가 아주 작거나 그 지령이 파일이름을 인수로서 받아 들이지 않을 때.
10. 함수 rd()는 아래에 보여 준것과 같다.

```
rd()  {
        dir  =  `pwd`
        cd  ..
        rm  -r  $dir
        }
```

11. . 또는 source지령을 사용하여 그 스크립트를 실행시킨다.
12. exec와 함께 .profile로부터 그 프로그람을 실행시킨다.

20장

1. 변수 x는 $x와 같이 씌여 져야 한다. 그것은 2^32 즉 4294967296을 현시한다는것을 의미한다.
2. perl -ne 'print "$.\t" . $_' foo를 사용한다.
3. 아래의 프로그람을 사용한다.

```
#!/usr/bin/perl -n
split(/:/) ;
print if $_[3] == 100 ;
```

4. perl -e 'print "UNIX" x 20 . "\n" ;'
5. perl -p -i -e "tr/[a-z]/[A-Z]/" foo
6. for순환을 사용한다.

```
#!/usr/bin/perl
print ("Enter a number: ") ;
$number = <STDIN> ;
if ( $number > 0 ) {
    for ( $x = 1 ; $x <= $number ; $x++ ) {
            print "$x\n" ;
    }
} else {
    print "Not a positive number\n" ;
}
```

7. 여기서 우리는 빈 문자렬(//)을 가지고 split를 사용한다.

```
#!/usr/bin/perl
print ("Enter a string: ") ;
$string = <STDIN> ;
chop ($string) ;
@arr = split (//, $string) ;
$length - @arr ;
for ( $x = 0 ; $x < $length ; $x++ ) {
    print "$arr[$x]\n" ;
}
```

8. 아래의 프로그람을 사용한다.

```
#!/usr/bin/perl
$number = 1 ;
while ( $number != 0 ) {
    print ("Enter a number: ") ;
    $number = <STDIN> ;
    chop ($number) ;
    if ( $number != 0 ) {
            $total += $number ;
    }
}
print "The total is $total\n" ;
```

9. &&와 ||연산자를 사용한다.

```
#!/usr/bin/perl
print ( "Enter a year: " ) ;
$year = <STDIN> ;
$remainder4 = $year % 4 ;
$remainder100 = $year % 100 ;
$remainder400 = $year % 400 ;
if ( ($remainder4 == 0 && $remainder100 !=
0) || ($remainder400 == 0) ) {
    print "It is a leap year\n" ;
} else {
    print "It is not a leap year\n" ;
}
```

10. 해석기행(1)에서는 !가, (2)에서는 "가, (3)에서는 ;이, (5)에서는 {가 빠졌으며 \n대신에 (6) /n이 사용되였다. 행번호들은 괄호안에 보여 주었다.
11. 아래의 프로그람을 사용한다.

```
#!/usr/bin/perl
print "String : ";
$a = <STDIN>;
print "Number of times : ";
chop ($b = <STDIN>);
$c = $a * $b;
print "The result is :  \n$c";
```

21장

1. /dev/fd0은 기정플로피구동기를 표현한다.
2. (1) /dev/fd0135ds18 & /dev/dsk/f0q18dt (SVR4)
   (2) /dev/fd0H1440 (Linux)

3. 확장구획은 몇개의 독립적인 론리구획을 포함한다. 따라서 하나의 확장구획안에 여러개의 파일체계를 가질수 있다.
4. 파일이름은 색인마디에 저장되지 않고 오직 등록부에 저장된다.
5. 련결계수기는 하나씩 감소된다.
6. 상위블로크안에.
7. /stand 나 /kernel (SVR4), /boot (Linux).
8. ls의 -i선택항목을 사용하는것으로.
9. 4개(IDE)와 8개(SCSI).
10. mkfs지령.
11. umount를 실행시키고 있는 사용자가 그 파일체계내부의 등록부에 위치하고 있거나 임의의 사용자가 그안의 파일을 열고 있을 때에는 내리우기 불가능하다.
12. fsck는 접속되지 않은 파일들을 /lost+found에 보낸다.
13. 오직 단일사용자방식에서만.

22장

1. /sbin과 /usr/sbin안에. 종전의 낡은 체계들은 /etc안에 가지고 있을것이다.
2. 상급사용자방식에서 calendar는 모든 사용자들의 달력파일을 찾아 내여 사용자에게 적당한 출력을 우편으로 보낸다.
3. /etc/shadow안에.
4. wall을 통보문내용과 함께 사용하는것으로써.
5. useradd -u 212 -g dialout -d /home/john -s /bin/bash -m john을 사용한다.
6. chsh지령으로(Solaris에서는 무효).
7. 핵심부는 unix나 genunix(SVR4) 그리고 vmlinuz(Linux)에 의하여 표현된다.
8. /etc/inittab의 action마당안에서 initdefault를 포함하는 행을 찾아 본다. 그 행에 지정된 실행준위가 기정실행준위이다.
9. [Ctrl][Alt][Del]을 사용한다.
10. shutdown -y -g0을 사용한다.
11. init는 체계를 특정한 상태(실행준위)로 유지하며 말단도구들에 getty프로쎄스들을 생성한다.
12. 실행준위 0은 체계를 끄며 상태 6은 체계를 재기동시킨다.
13. 할수 없다. lpsched는 단일사용자방식에서 작업하지 않기때문에 불가능하다.
14. telnet q 또는 init q를 사용하여 init가 inittab를 다시 읽게 한다.

23장

1. (1) C클라스. (2) A클라스. (3) A클라스.
2. 처음것은 가질수 있으나 둘째것은 인터네트에 예약된것이다.
3. IRQ와 I/O주소.
4. ifconfig -a를 사용한다.
5. /etc/inetd.conf안에서 ftp에 속하는 행에 주해를 주는것에 의하여(commenting)
6. PPP(점대점통신규약). pppd지령이 그 봉사를 호출한다.
7. ATDT(음성접속) 또는 ATDP(임풀스접속)를 사용한다.
8. /etc/exports는 원격으로 접속될수 있는 등록부들을 보여 주며 의뢰기의 접근권한들을 지정한다. exportfs -a 지령을 사용함으로써 변화된 정보가 mount데몬에 유효하게 되도록 할수 있다. 지령이 유효하지 않다면 rcn.d등록부안에 있는 적절한 《시작》스크립트를 restart인수와 함께 호출하여야 한다.

24장

1. 종속봉사기는 주봉사기의 여벌용으로서 봉사하며 지역전송을 통하여 주봉사기로부터 자료를 얻는다.
2. ftp는 /etc/resolv.conf로부터 이름봉사기등록들을 읽어 내는 해석기를 통하여 동작한다.
3. MX레코드는 우편봉사기들의 주소와 그의 우선권을 지정하는데 사용된다. 낮은 우선권번호를 가진 봉사기가 더 높은 번호를 가진 봉사기보다 먼저 시도된다.
4. CNAME레코드는 주콤퓨터이름에 관한 별명들을 정의하는데 리용된다.
5. 우편배포대행체(MDA)는 SMTP로부터 우편을 가져 오며 배포를 수행한다. mail, deliver, procmail이 일반적인 우편배포프로그람들이다.
6. 우편은 /var/spool/mqueue안에 완충되며 그 목록은 mailq지령으로 현시한다.
7. 파일 /etc/sendmail.cw안에 모든 령역들을 정의하고 그 파일을 Fw/etc/sendmail.cw로 정의한다.
8. telnet hostname 25와 telnet hostname 110을 사용하고 통보문을 얻는다.
9. 그 파일을 newaliases로 콤파일하지 않았다.
10. httpd.conf안의 MinSpareServers에 의하여 결정된 수만큼.
11. DocumentRoot명령으로

부록 9. 참고문헌

일반열람

The UNIX Programming Environment, by Brian Kernighan and Rob Pike (Prentice Hall), 1984, Englewood Cliffs, New Jersey.

The Design of the UNIX Operating System, by Maurice Bach (Prentice Hall), 1986, Englewood Cliffs, New Jersey.

The UNIX Operating System, by Kaare Christian (John Wiley), 1988, New York.

Open Computing UNIX Unbound, by Harley Hahn (Osborne McGraw-Hill), 1994, Berkeley, California.

UNIX System V: A Practical Guide, 3rd Edition, by Mark G. Sobell (Addison-Wesley), 1995, Menlo Park, California.

UNIX System V Release 4: An Introduction, 2nd Edition, by Kenneth Rosen, Richard Rosinski & James Farber (Osborne McGraw Hill), 1996, Berkeley, California.

UNIX Power Tools by Jerry Peek, Tim O'Reilly and Mike Loukides (O'Reilly and Associates), 1994, Sebastopol, California.

Open Computing's Best UNIX Tips Ever by Kenneth Rosen, Richard Rosinski and Douglas Host (Osborne McGraw-Hill), 1994, Berkeley, California.

Running Linux, Third Edition, by Matt Welsh, Matthias Kalle Dalheimer and Lar Kaufman (O'Reilly & Associates), 1999, Sebastopol, California.

Using Linux, Fifth Edition, by Jack Tackett and David Gunter (Que Corporation), 2000, Indianapolis, Indiana.

Red Hat Linux 6 Unleashed by David Pitts, Bill Ball et al. (Sams Publishing), 1999.

Using UNIX, Special Edition, by Peter Kuo (Lead Author) (Que Corporation), 1998, Indianapolis, Indiana.

주제별 열람

A Guide to vi by Dan Sonnenschein (Prentice Hall), 1987, Englewood Cliffs, New Jersey.

GNU Emacs-UNIX Text Editing and Programming by M. Schoonover, J. S. Bowie and W. R. Arnold (Addison-Wesley), 1992, Reading, Massachusetts.

Learning GNU Emacs, 2nd Edition, by Debra Cameron, Bill Rosenblatt and Eric Raymond (O'Reilly & Associates), 1996, Sebastopol, California.

UNIX Application Programming by Ray Swartz (SAMS), 1990, Carmel, Indiana.

Learning the Korn Shell by Bill Rosenblatt (O'Reilly & Associates), 1994, Sebastopol, California.

Learning the bash Shell by Cameron Newham and Bill Rosenblatt (O'Reilly & Associates), 1998, Sebastopol, California.

Programming Perl by Larry Wall and Randal Schwartz (O'Reilly & Associates), 1991, Sebastopol, California.

Learning Perl by Randal Schwartz (O'Reilly & Associates), 1993, Sebastopol, California.

Teach Yourself Perl 5 in 21 Days by David Till (SAMS Publishing), 1996, Indianapolis, Indiana.

Perl 5 Interactive Course by Jon Orwant (Waite Group), 1996.

Teach Yourself CGI Programming with Perl 5 in a Week by Eric Herrman (Sams.net Publishing), 1997.

The X Window System User's Guide-Motif Edition by Valerie Quercia and Tim O'Reilly (O'Reilly & Associates), 1993, Sebastopol, California.

TCP/IP망과 인터네트

TCP/IP Network Administration by Craig Hunt (O'Reilly & Associates), 1998, Sebastopol, California.

Linux Network Servers by Craig Hunt (Network Press), 1999, Alameda, California.

DNS and BIND, Third Edition, by Paul Albitz and Cricket Liu (O'Reilly & Associates), 1998, Sebastopol, California.

sendmail by Bryan Costales with Eric Allman (O'Reilly & Associates), 1997, Sebastopol, California.

Internet Secrets by John Levine and Corol Baroudi (IDG Books Worldwide), 1996, Foster City, California.

The Whole Internet-User's Guide and Catalog by Ed Krol (O'Reilly & Associates), 1994, Sebastopol, California.

Harley Hahn's The Internet Complete Reference, Second Edition, by Harley Hahn (Osborne McGraw-Hill), 1996, Berkeley, California.

체계관리

UNIX System V Release 4 Administration by David Fiedler and Bruce Hunter (Hayden Books), 1995, Indianapolis, Indiana.

UNIX System Administration Handbook by Evi Nemeth, Garth Snyder, Scott Seebass and Trent Hein (Prentice Hall PTR), 1995, Upper Saddle River, New Jersey.

Essential System Administration by AEleen Frisch (O'Reilly & Associates), 1995, Sebastopol, California.

UNIX System Administrator's Bible by Yves Lepage and Paul Iarrera (IDG Books Worldwide), 1998, Foster City, California.

UNIX System Administrator's Edition by Robin Birk and David B.Horvath et al. (Sams Publishing), 1997.

System Performance Tuning by Mike Loukides (O'Reilly & Associates), 1992, Sebastopol, California.

잡지와 론문

"The UNIX Time-Sharing System" by Dennis Ritchie and Ken Thompson (Bell System Technical Journal, Vol. 57, No.6), 1978, Murray Hill, New Jersey.

"UNIX Implementation" by Ken Thompson (Bell System Technical Journal, Vol. 57, No.6), 1978, Murray Hill, New Jersey.

"A Retrospective" by Dennis Ritchie (Bell System Technical Journal, Vol. 57, No.6), 1978, Murray Hill, New Jersey.

"The UNIX Shell" by Steve Bourne (Bell System Technical Journal, Vol. 57, No.6), 1978, Nurray Hill, New Jersey.

Awk-A Pattern Scanning and Processing Language Programmer's Manual by Alfred Aho, Peter Weinberger and Brian Kernighan (Computing Science Technical Report No.118 of AT&T Bell Labs), Murray Hill, New Jersey.

색 인